D[illegible]
N[illegible]
M[illegible] CALIFORNIA 93943

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

THE SIMULATION OF A MAJOR AUTOMATED INFORMATION SYSTEM (AIS) ON A MICROCOMPUTER

by

Keith V. Lockett
Michael E. O'Neil

March 1984

Thesis Advisor: Joseph F. Mullane,Jr.

Approved for public release; distribution unlimited.

T213250

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): The Simulation of a Major Automated Information System (AIS) on a Microcomputer	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis March 1984
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Keith V. Lockett, Michael E. O'Neil	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, CA 93943	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, CA 93943	12. REPORT DATE: March 1984
	13. NUMBER OF PAGES: 187
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Microcomputer, Automated Information System, Simulation, Simulation Development Lifecycle, Software Decision Model, Hardware_Evaluation Matrix, Test and Evaluation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The objective of this thesis is to determine if a mainframe computer Automated Information System (AIS) can be simulated on a conventional microcomputer. To this end, the topical areas of software, hardware, the simulation development lifecycle, and systems testing and evaluation are explored in-depth. The purpose of this in-depth subject area examination is to demonstrate the tradeoffs and decision points encountered in the systems management lifecycle. Recommendations based upon these tradeoffs and decisions are then presented. Lastly, the conclusions address the attainment of the thesis objective.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Approved for public release, distribution unlimited

The Simulation of a Major Automated Information System (AIS) on a Microcomputer

by

Keith V. Lockett
Captain, United States Marine Corps
B.S., United States Naval Academy, 1977

and

Michael E. O'Neil
Captain, United States Marine Corps
B.S., Miami University, 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March 1984

ABSTRACT

The objective of this thesis is to determine if a mainframe computer Automated Information System (AIS) can be simulated on a conventional microcomputer. To this end, the topical areas of software, hardware, the simulation development lifecycle, and systems testing and evaluation are explored in-depth. The purpose of this in-depth subject area examination is to demonstrate the tradeoffs and decision points encountered in the systems management lifecycle. Recommendations based upon these tradeoffs and decisions are then presented. Lastly, the conclusions address the attainment of the thesis objective.

TABLE OF CONTENTS

I. INTRODUCTION 7

II. BACKGROUND 11

A. PURPOSE 11

B. AUTOMATED INFORMATION SYSTEMS 11

C. THE MICROCOMPUTER 22

D. SIMULATION 30

III. PROBLEM DEFINITION AND INITIAL DECISION PHASE 44

A. REVIEW 44

B. PROBLEM STATEMENT 44

C. DETERMINATION OF THE AIS TO BE SIMULATED ... 48

D. THE USER GROUP 53

IV. SOFTWARE 57

A. OVERVIEW 57

B. DEFINITION OF SOFTWARE REQUIREMENTS 58

C. THE SOFTWARE DECISION MODEL 68

D. THE SOFTWARE ALTERNATIVES 71

E. ALTERNATIVE EVALUATION AND SOFTWARE SELECTION USING THE SOFTWARE DECISION MODEL 81

V. HARDWARE 84

A. INTRODUCTION 84

B. HARDWARE SELECTION AND EVALUATION 85

C. HARDWARE SELECTION/EVALUATION DEVELOPMENT TEAM ADAPTATION 89

D. ACQUISITION 93

E. ACQUISITION - DEVELOPMENT TEAM ADAPTATION .. 96

F. IMPLEMENTATION 99

G. IMPLEMENTATION - DEVELOPMENT TEAM ADAPTATION 105

H. MAINTENANCE 108

I. MAINTENANCE - DEVELOPMENT TEAM ADAPTATION ..110

VI. THE SIMULATION DEVELOPMENT LIFECYCLE 112

A. INTRODUCTION 112

B. THE SIMULATION FEASIBILITY PHASE 113

C. THE SIMULATION REQUIREMENTS AND SPECIFICATION PHASE 115

D. THE CONCEPTUAL SIMULATION DESIGN PHASE 118

E. THE DETAILED SIMULATION DESIGN PHASE 121

F. THE SIMULATION MODULE CREATION PHASE 122

G. THE SIMULATION MODULE INTEGRATION PHASE 124

H. THE SIMULATION IMPLEMENTATION PHASE 125

I. THE SIMULATION MAINTENANCE PHASE 127

VII. TEST AND EVALUATION 128

A. INTRODUCTION 128

B. BACKGROUND 128

C. FORMAL AND INFORMAL TESTING PROCEDURES 130

D. REQUIREMENTS SPECIFICATIONS, TESTING AND QUALITY ASSURANCE 135

E. CHARACTERISTICS OF QUALITY SOFTWARE 139

F. QUALITY METRICS 141

G. AUTOMATED TOOLS THAT AID TESTING AND EVALUATION 141

H. LIFECYCLE IMPLICATIONS OF TESTING AND EVALUATION 145

I. PEP SIMULATION TEST AND EVALUATION RESULTS 147

VIII. CONCLUSIONS AND RECOMMENDATIONS 152

A. INTRODUCTION 152

B. RESEARCH QUESTION RESPONSES 152

C. CONCLUSIONS 159

D. RECOMMENDATIONS 164

APPENDIX A : The Software Decision Model 168

APPENDIX B : The Hardware Evaluation Matrix 170

APPENDIX C : PEP Fundamental System Model 173

APPENDIX D : PEP Hierarchial Process Diagram 174

APPENDIX E : Subsystem and Program Module Decomposition Summary 175

APPENDIX F : PEP Microcomputer Simulation User Group Survey 176

APPENDIX G : Analysis of MCPCC Survey 179

LIST OF REFERENCES 184

INITIAL DISTRIBUTION LIST 186

I. INTRODUCTION

In any decision making environment, a manager has three primary variables: manpower, money and information. Manpower, while still fundamental to any decision, enjoyed it's major emphasis in decision making prior to the industrial revolution. With the onset of the industrial revolution, money became the dominant decision variable. As we move toward the 21st century, it is becoming obvious to many that manpower and money are becoming subordinate to information in the decision maker's environment. [Ref 1 : pp. 11 - 38]

The emergence of information as the dominant decision variable is the result of several trends. The first trend is the fact that information permeates both money and manpower, as it tends to encapsulate the salient characteristics of both. Secondly, the increasing speed of information flow has tended to cause it to be the overbearing factor in many decision situations. This information flow intensification has largely been the byproduct of computer and communications technology, coupled with an exponential growth in media forms. Finally, the organizational environment has undergone an 'information explosion'. This explosion is the result of the aforementioned speed increase of information, and an

unprecedented growth in the absolute quantity of information confronting decision makers. [Ref 1 : pp. 189 - 205]

Recognizing the dominance of information to decision makers, it is the intent of this work to concurrently examine and combine three vital information subsets: automated information systems, microcomputers and simulation. To tackle this triad, a seven step/chapter format will be employed.

To begin this process, background information on automated information systems, microcomputers and simulation will be presented. This will provide readers with a common knowledge base on these three topics.

Following will be the problem definition. Included in this chapter will be the relevant research questions, and a descriptive explanation of the initial decisions surrounding this thesis.

Next, a decision model for microcomputer simulation software will be introduced. This decision model will reflect the software requirements and specifications of this thesis, and is designed to serve as an objective decision aid to the authors.

Complimenting the software decision model will be a chapter on microcomputer hardware. Topics covered in this chapter include the selection process, the acquisition procedures and the maintenance aspects of the hardware devices.

The next chapter will cover the development of a simulated automated information system on a microcomputer. Relying on the previous software and hardware decisions, the topic is the focal theme of this thesis. Critical to this development process is the simulation lifecycle, which serves a structural framework for the process and a decision aid for the development team.

A subset of the simulation lifecycle, the test and evaluation phase, is the subject of a separate chapter. Testing and evaluation is given separate emphasis, as it represents the critical link between the requirements and specifications and the delivered product. This topic comprises unit testing, systems testing and the user feedback loop.

The last chapter serves to summarize the entire work through recommendations and conclusions. Recommendations address possible future directions of this thesis. The conclusions will provide answers to the previously introduced research questions.

Explicitly, this thesis deals with numerous, intertwined technical topics. Underlying these surface areas are the fundamental topics of communication and decision tradeoffs. Throughout this work, every effort will be made to jointly couple these fundamental subject areas with the technical themes. This deliberate emphasis is intended to illustrate

the multi-dimensional environment in which this thesis was developed and exists.

II. BACKGROUND

A. PURPOSE

In order to evaluate the researchable questions, it is necessary to describe and analyze the three major subsets of this thesis: Automated Information Systems, Microcomputers, and Simulation. This chapter will develop a historical and academic perspective of these three topics.

B. AUTOMATED INFORMATION SYSTEMS

A manager confronts his crowded desk. The face of his desk is layered with memorandums, correspondence and reports. Among the reports are several printouts from the data processing center. These particular reports, printed entirely in upper case by a line printer, are the byproducts of an Automated Information System (AIS). The term Automated Information System describes the information resource manager in most organizations.

An AIS typically is able to accumulate data from multiple locations. Collection mechanisms include data entry at cathode ray tube (CRT) terminals by human operators, Hollerith cards, magnetic tapes, bar code readers, remote job entry, and many types of mechanical and electronic sensing devices. Once the raw data has been routed into an AIS channel, it normally undergoes a series of edits to validate and verify the content. The AIS

editing process is intended to prevent the 'Garbage In, Garbage Out' cycle of information processing. Upon edit cycle completion, valid data enters into the processing cycle. The purpose of the processing cycle is to aggregate data into an organizational resource for decisions: information.

The preceding description emphasizes the information development and production phases of an AIS. In addition, an AIS normally serves as an organization's information storage medium. As with data collection, a diverse mixture of storage devices are employed e.g. disk packs, magnetic tape, hard copy printout and microfiche. Prevalent throughout both the production and storage functions of an AIS are the security features. These features limit access to information upon organizationally defined criteria. Security of an AIS is a dynamic and sensitive area, as the AIS frequently contains undisclosed elements of corporate strategy.

In addition to producing and testing information, an AIS communicates with the organization. This communication is accomplished through the channeling of information to different organizational echelons. Although no universal structure or format for this channeling exists, it can generally be categorized as follows:

1. Scheduled Reports: Normally produced in hard copy printout on a recurring basis. Primarily employed by operational/tactical level managers as a barometer of day-to-day operations and performance.

2. Unscheduled Reports: Normally produced in hard copy printout only when an exception condition occurs. Primarily used by middle/control level managers as indicators of potential or existing problem areas.

3. Inquiry Responses: Normally produced in a real time environment via soft copy on a CRT. Used by upper/strategic level managers as a source of information to aid in decision making and/or in planning.

The presentation of these categories strongly indicate that the output of an AIS is intended to directly affect managerial decision making. Herein lies the objective of any AIS: To provide timely, accurate and understandable information to users designed to foster the accomplishment of organizational objectives and goals.

An AIS does not exist in a vacuum. Rather, it interfaces with several different environments. The two primary AIS interfaces are with the hardware/software environment in which the AIS dwells, and the larger organizational environment which the AIS supports.

In physical terms, an AIS is merely a complex and integrated coupling of hardware and software components. This coupling causes the AIS to be perceived as a transparent mechanism to the external organization, except to the data processing department. Routinely, the hardware foundation of an AIS is a large, general purpose mainframe computer. This hardware foundation reflects the early origins of centralized data processing in many concerns. Additionally, it indicates an organization's desire to control the access and distribution of information.

Examples of such general purpose mainframe computers are the IBM 30XX series, and the Amdahl 470/VX series.

The dynamics of technology have caused the hardware arena to expand outward in several directions. The first outward step involved the hard wiring of terminals to a central mainframe computer. The appearance of such terminals, like the IBM 327X series and the Telex 278-2, reflect the trend toward interactive data entry and inquiry use of an AIS. The logical successors to 'dumb' terminals such as these are the current generation of 'smart' terminals. Possessing all the capabilities of dumb terminals, a smart terminal also is capable of editing and performing some preprocessing of data prior to transmission to a central mainframe processor. The smart terminal positions a significant amount of computing power directly at the user level, while simultaneously reducing the processing workload of the mainframe. The best illustration of a smart terminal is the field salesman with his microcomputer and modem. He can input and prepare his reports on his microcomputer at a decentralized location, and then transmit his preprocessed data to the central information manager via normal telephone links.

An alternative architecture to the smart terminal approach is to use dumb terminals coupled to a front-end processor. The front-end processor is normally a minicomputer, which is dedicated to editing and limited

processing functions before transmitting data packages to the central information manager. The DEC PDP-11 is frequently used as a front-end processor. This approach to the hardware side of an AIS is particularly well suited when data is transmitted from multiple locations, the data is edit intensive, or authority/responsibility is decentralized to the regional organizational levels.

Several AIS hardware alternatives are currently developing on the leading edge of technology. Heightened interest and intensive research is revolving around the database machine. The database machine is a back end processor ; it logically lives behind the central information manager. It is functionally and physically dedicated to database operations. Such devices are architecturally tailored to the unique aspects of database operations, e.g. specially designed memory registers, and the application of expandable buffers. It is likely that the database machine will gain a strong following in the AIS world, as they represent an economical and more secure answer to the shrinking availability of mainframe processing resources.

Another area of interest involves distributed AIS systems. Conceptually, such a system will allow the structuring of an AIS, both in logical and physical terms, to directly correspond to the organizational responsibility/authority levels. Currently, the distributed

system used internally by Hewlett Packard in support of sales, manufacturing and distribution operations represents a successful distributed AIS. The benefits of such a distributed AIS are stronger intra-organizational information flow, and enhanced real time information. Organizations have been fairly reluctant to venture into the distributed AIS arena. This reluctance is a function of several highly publicized distributed AIS failures, e.g. IBM's failure to develop a distributed reservation and reporting system for the Holiday Inn Corporation in the early 1970's. The lack of concrete corporate information strategy and an inherent organizational tendency to centralized information processing functions has also slowed corporate entry into the distributed arena.

Complimenting the hardware functions of an AIS are powerful software systems. The basic AIS software package normally incorporates the following functions:

1. Transaction Processing: The input, editing and appending of transactions to files and/or databases.
2. File/Database Management: A file or database subsystem, structured in hierarchical , network or relational architecture.
3. Report Generator: An output oriented function, designed to assemble information from multiple sources into user-readable formats.

The above features are normally layered several tiers above the operating system in the virtual machine. Dependent upon the respective AIS source language, the operating system can serve varied functions. In a COBOL-based AIS, the operating

system may serve as the supporting file manager. Increasingly, many AIS's are being created with modern, powerful, pseudo-procedural packages known as fourth generation lnaguages. Information Builder's FOCUS and Mathematica's RAMIS II are current examples of DBMS-oriented, fourth generation languages. These fourth generation languages may ignore the host operating system and establish its own physical interface with the hardware. They are also fully capable of being piggybacked on top of many existing operating systems and database management systems. Regardless of the circumstances, the presence of the operating system is normally transparent to the AIS user.

Initially, COBOL was a nearly universal choice for the AIS source language. The extensive use of COBOL reflected the government's heavy sponsoring and backing of the language, and the semi-structured nature of the language. As the complexity and use of AIS's grew, COBOL has become less common as a source language. This decline in the use of COBOL in new systems comes from COBOL's orientation to batch processing, it's lack of organic statistical capability and the appearance of integrated database management packages. However, COBOL is by no means a bygone memory. To the contrary, the multi-million line COBOL-based AIS's present throughout the public and private sector will ensure the existence of COBOL well into the next century.

As previously described, hardware technology has been changing at an unprecedented rate. Lagging behind the hardware revolution has been the software evolution. Relative to the AIS world, the software evolution is perhaps best illustrated by the proliferation of database management systems (DBMS). IBM's relational database SEQUEL, and Software AG's network database ADABAS are two currently successful commercial examples. The integration of a DBMS into an AIS frequently serves to extend the useful life of an AIS. Also, it permits the evolution or expansion of an AIS from the batch processing environment to the interactive processing environment. Lastly, it produces mixed results by broadening the base of an organization's AIS users.

Outside the hardware/software environment that an AIS exists in is the broader organizational environment. It is in this environment that an AIS has several areas of significant concern. First, the AIS is usually the primary responsibility and source of corporate identity for the data processing department. This routinely generates organizational conflict, especially with the growth of interactive processing. Secondly, an AIS normally parallels the reporting and performance evaluation hierarchy within an organization. If improperly used, subordinates can fall into a numbers game to satisfy the AIS scoreboard and lose sight of the underlying organizational objectives and goals. Also, the AIS can develop into the primary communication

link within an organization. Generally, this is improper as the AIS is intended to enhance decision making and not circumvent human communication. The prevalence of AIS's in organizations has nurtured the development of 'management by exception' leadership styles in certain managerial circles. While such leadership styles have a definite time and place in any time-parametered organization, it's value as a blanket replacement for the more traditional leadership styles is debatable. The implementation of an AIS within an organization normally produces an avalanche of information at the decision maker level. While permitting more informed decisions, an AIS can also serve to cloud a manager's perspective with mountains of trivia. Lastly, AIS's have significantly increased the speed of information flow within organizations. Given reliable computer resources, much of the timelag and information 'float' present in previous manual systems has vanished. In the face of computer resource failure, information can be inaccurate, unavailable or simply vanish.

As is evident by the preceding paragraphs, an AIS is a complex mechanism with significant and far reaching organizational effect. A classic real world example of a dynamic and evolving AIS is the Joint Uniform Military Pay System (JUMPS), the uniformed services compensation system employed by DOD. Another public sector AIS is the Supported Activity Supply System (SASSY), the supply system employed

by the operating forces of the U. S. Marine Corps. In the private sector, the centralized corporate management reporting and information system employed by Texas Instruments is an example of a successful corporate AIS.

Like any evolving technology, AIS's are a two-edged sword. Many pitfalls surround the implementation and use of an AIS. Perhaps the biggest drawback is that any AIS is highly susceptible to Parkinson's Law. That is, the workload and information requirements of an AIS will tend to expand to equal the available resources. The net result of this effect is a scarcity of mainframe processing resources. A public sector example of this phenomenon is the U. S. Marine Corps. Having recently purchased seven (7) Amdahl 470/V7 mainframe processors, the Marine Corps felt that it's processing requirements would be satisfied until the mid-1990's. This proved otherwise, as this organization is currently experiencing a 92-95% CPU utilization rate [Ref. 2 : pg. v-1].

A second fundamental AIS problem deals with the primary output of an AIS: information. Information can lack value, lack timeliness, lack validity and experience quality and quantity weaknesses. The lack of timeliness may be a function of system output scheduling, age of data and timelag of input or more simply, the steadfast requirement to have real-time information in certain decision making situations. The timeliness obstacle is diminishing with the

growth of interactive processing. Validity problems can be driven by a myriad of problems ranging from system logic design errors to intentionally erroneous data input. Finally, an AIS is a vast reservoir of information. The output of this reservoir can flood the user with data. The value of this data, both from a quality and quantity perspective, in actual decision making and problem solving is often suspect. This quality/quantity problem is primarily present at the operational levels of management.

The other primary problem of an AIS revolves around the system development. In the system development arena, the problems stem from interfaces, conversion of systems and the training/documentation/maintenance phase of the AIS lifecycle.

Interface problems develop both at the user level, and with the existing and external systems. User level interface faults surface around the organizational interface with users, definition of user requirements, and user communication channels. Interfacing with existing and external systems causes data compatibility, telecommunications and hardware compatibility questions to be raised.

The most exasperating and time intensive system development dillema relates to the conversion of existing systems. A broad problem spectrum, including data structure considerations, source code transformation, and DBMS physical structure decisions, must be addressed when

converting an AIS. The best tools to have when dealing with conversion problems are an experienced and open minded AIS manager, and established corporate information policy. The remaining basic system development dilemma involves the ongoing training/documentation/maintenance phase of the AIS lifecycle.

Training frequently spells the difference between success and failure of an AIS in an organization. The ultimate performance barometer is user satisfaction. Documentation is an expensive ongoing and iterative process that can be significantly reduced by effective management. Complete and accurate documentation is essential, as it is the historical and permanent guide to AIS development. This is critical in the AIS arena, as personnel turnover is routinely rapid. AIS can produce synergistic difficulties. The way to avoid these difficulties is well planned maintenance in response to user requirements and resource availability constraints, coupled with a clearly defined corporate information policy.

C. THE MICROCOMPUTER

More recently, a new character has crept into the spectrum of managerial decision making. Hailing from a portion of California known as the 'Silicon Valley', this new decision partner is the microcomputer. With curious names like the Apple II and Heathkit, managers initially rebuffed the microcomputer as a hobbyist's toy. The cottage

microcomputer industry listened to the message, as their product offerings underwent an exponential propagation in quantity and a definitive increase in quality and reliability. The best example of such a 'refined' product offering was the Apple II+ microcomputer coupled with the VISICALC electronic spreadsheet. This point of departure, in the late 1979 - early 1980 timeframe, marked the migration of the microcomputer from the hobbyist's workbench to the manager's desktop. Later events, ranging from IBM's introduction of their Personal Computer in September, 1982 to the development of integrated management software tools like Lotus 1-2-3 and Context MBA, have firmly established the microcomputer as a managerial decision aid. The microcomputer entry into the management limelight, coupled with their relatively low cost, has led to a proliferation of microcomputers in many organizations.

With this inordinate amount of interest generated towards microcomputers, it becomes necessary to examine the 5 W's and the How of the microcomputer. Simply stated, a microcomputer is an electronic device that accepts data, manipulates it to solve problems and produces an organizational resource: information [Ref. 3 : p. 40]. This electronic device, like any computer, is composed of six interrelated components:

1. The Input Device: A mechanism that allows the transformation of raw data into a format understandable by the computer. In microcomputer systems, the standard input device is a keyboard

coupled with a video monitor/cathode ray tube (CRT). This allows the operator to see and verify the input.

2. The Memory: The internal data warehouse of the computer. It allows random access of data, and is normally called RAM (random access memory). RAM capability is measured in kilobytes (KB). A kilobyte equates to the storage of 1000 arabic characters. At a minimum, a business oriented microcomputer system should possess 64 KB of RAM, with 128 KB or greater preferred.

3. Secondary Storage: The external data warehouse of the computer. Depending on the microcomputer system, secondary storage devices can be floppy diskettes, magnetic hard disks or cassette tapes. Access to data is relatively longer than RAM, but more rapid than comparable manual means. The capability of secondary storage is also measured in KB's. This capability varies tremendously. As such, no universal 'rule of thumb' exists for secondary storage capacity. Rather, secondary storage capacity should be directly reflective of the needs of the respective application.

4. The Central Processing Unit (CPU): This is the internal brain of the computer. The CPU thinks entirely in numeric terms, and its ability to think as such provides performance measures. Often, a CPU is spoken of in terms of 'clock speed', which is stated in megahertz (MHZ). In plain language, clock speed is the rate at which the CPU thinks and operates. A faster clock will produce quicker results than a slower clock, all other things equal. Microcomputer clock speeds range from 1.5 MHZ to greater than 10 MHZ. CPU's are also measured in terms of 'word size'. Word size is the number of binary characters (bits) used by the CPU to communicate within itself and with other parts of the computer. A bigger word size will serve to increase the speed at which arithmetic and logical operations are performed, and produces increased numeric accuracy than a smaller word size. Typical word sizes for microcomputers are 8, 16 and 32 bits.

5. The Output Device: This is the medium used to present information from the computer to the user. Output devices are normally either CRT's and/or printers. CRT's are measured in terms of screen resolution, and screen size. Generally, a 9 inch diagonal CRT with 160 by 100 pixel resolution is the minimum to be considered for a microcomputer system. Also, CRT's are available in numerous screen colors. Amber is

thought to be the most favorable color for human users, with green phosphorus also being used widely. Multi-color screens, known as RGB for their use of the primary colors red, green and blue to produce the full spectrum of colors and hues, are available and frequently used in training environments. Likewise, printers can be thought of in two ways: print readability, either letter quality or dot matrix, and speed, in characters printed per second. The printer to be used is a device tradeoff between the desired speed of print and the application e.g. word processing, forms, graphics.

6. The Software: This is the real power of any computer. It gives the CPU common sense and applicability from a user's perspective. Additionally, it serves as the interface layer between the bare electronic machine and the user's keystrokes on the keyboard. It comes in a multitude of varieties and flavors, and can be considered to be somewhat specialized. Owing to this specialization, the software should be the first and primary component selected in any microcomputer system.

These six components, when bundled into a coordinated package, possess the ability to enhance productivity in a many decision environments. The critical caveat to any package is that a clearcut requirement/specification should drive the package configuration, rather than the package configuration being the result or reflection of cleverly projected advertising campaigns.

As might be anticipated, microcomputers have significantly affected organizations. The direction and magnitude of the microcomputer ripple effect is permeating into all levels of organizations, ranging from sole proprietorships to the boardrooms of the Fortune 500. Perhaps the foremost microcomputer impact has been the tendency to decentralize organizational power and knowledge

bases. Whereas a middle level manager would have had a staff analyst to aggregate and interpret information, now a subordinate manager can perform the same tasks with a microcomputer. Many organizations have done some real soul searching to deal with this sudden and unplanned migration of information. A related spinoff to this migration has been the reduction in staff analysts in many organizations [Ref. 4: p. 68]. Microcomputers are being economically substituted for these staff analysts, creating a leaner organization and a cadre of unemployed staff analysts. Another affect of microcomputers in many organizations has been the demystification of much of the aura and black magic previously engulfing computers. Simultaneous with this demystification has been the dilemma of weakened functionality and vanishing corporate identity for data processing (DP) departments. What had formerly been the exclusive domain of DP personnel has now ballooned to include any manager with a microcomputer and modem. Expectedly, this situation has generated intra-organizational conflict. No simple solution to resolve this conflict has proven optimal. However, the presence of concrete organizational information policy has tended to minimize this type of conflict [Ref. 5: pg.42]. A more predictable result of microcomputers in organizations has been the development of data redundancy, appearance of many data security problems and the emergence of an entire

spectrum of compatibility considerations. Data redundancy has been reduced by the appearance of multi-user, shared hard disks. Security, however, is a multi-dimensional issue, with no absolute solution on the horizon. The compatibility question is resolvable with solid, defined corporate information policies and procedures.

Another effect of microcomputers has been the tendency of some users to divert energies away from their assigned tasks due to a compulsive interest in the computer. This compulsion goes beyond learning the applications programs of the microcomputer, and can effectively turn a manager into a assembly language programmer. This is a serious negative result, and can result in non productive expenditure of time and resources. The best solution to this dilemma is the traditional role of the manager as a supervisor of his subordinates, to insure that a subordinate is investing adequate resources into his primary responsibilities and not evolving into a microcomputer addict.

Lastly, the entry of microcomputers into an organization often creates a new status symbol and power emblem. This is a potentially dangerous situation, as it represents a source of intra-organizational conflict and a possible resource expenditure for pure status purposes, not productivity.

The description of microcomputers in organizations described herein creates a Pandora's box perception. Despite this portrayal, microcomputers have had a positive

and advantageous effect on many organizations. Perhaps the most visible and relevant effect has been the enhanced productivity and additional time availability enjoyed by decision makers. The mere presence of an automated 'What If' capability is much more time efficient and accurate than the time honored stubby pencil and ledger pad. The combined impact of productivity and time availability is particularly germane when dealing with crisis management situations, the plague of operational and control level managers. An example of practical use of this 'What If' capability involves the recalculation of pre-exercise cost estimates in the U. S. Marine Corps. These estimates, which are usually driven by several key variables like type and quantity of aircraft flight hours, tank operating hours, amphibian assault vehicle operating hours and battalion field training days, can readily be modeled using an electronic spreadsheet. To recalculate the total cost estimate using such an electronic spreadsheet involves the simple changing of several total resource inputs, with the reminder of the calculation being done by the microcomputer. This procedure would take an intermediate computer user several minutes. Manual recalculation of the same exercise cost estimate is usually a several hour, if not several day, process, and is subject to a great variety of errors.

The presence of microcomputers has also created the phenomenon known as 'silent firing'. [Ref. 6: pg. 28]

Silent firing involves the application and exploitation of a microcomputer to handle an increasing workload, vice increasing the personnel staff. This can be an extremely resource efficient usage, especially in the face of rising personnel costs.

Microcomputers have also allowed users to get multiple and diverse perspectives of the same information. To illustrate such a situation, a DBMS will permit numerous views of a consumer demographics database. These various views can often unlock the secret to penetrating a designated target market, or enhancing the promotional effectiveness within a given Standard Metropolitan Statistical Area (SMSA).

The microcomputer graphics revolution has bolstered intra-organizational communications. To wit, the desire to have graphics capability by activity level comptrollers in the U. S. Marine Corps is an instance of such communication [Ref. 7: pg. 30].

Finally, when properly implemented, microcomputers can be a significant morale booster within an organization. Organizational members frequently feel they are part of the 'high tech metamorphosis' that seems to be enveloping society. Being a part of what is perceived as a progressive and affirmative movement normally has a beneficial effect on group members.

D. SIMULATION

To simulate generally denotes performing actions that tangibly resemble reality. With regard to decision making, simulation normally means the use of a support environment e.g. computers, visual aids to interact with and exercise a model of a parent system. With simulation, the nature of the outcomes is not preordained upon interaction. Rather, a full spectrum of resultant conditions is possible. To summarize these descriptions, simulation can be conceptualized as an interactive participant's tool box. The tool box contains multiple models and a legion of conceivable consequences.

Shannon defines simulation as the process of designing a model of a real world system and conducting experiments with the model [Ref. 8: pg. 2]. The purpose of these experiments is to gain an understanding of the system and its related behavior, and to analyze various strategies (within known and artificial constraints) for the operation of the parent system.

Despite the generality of this definition, it is possible to identify several intrinsic characteristics of simulation. Foremost is the trait that simulation is driven by models. Second, simulation is a descriptive, not a normative, management device. Lastly, simulation is used when the parent system is too obtrusive to be encapsulated by routine analytical and optimization methods.

The relationship between simulation and models is essential. Models serve as the building blocks to represent and parallel reality. By constructively organizing these building blocks, a simulation is created. The use of the building block concept permits modular development of simulations, and favors graphical representation of simulations. A well conceived model attempts to incorporate only the relevant and accurate aspects of reality, and desires to avoid irrelevant aspects. This modeling feature tends to decrease the complexity of simulations, increases their resource efficiency and simplifies user understanding of the simulation.

By being a descriptive device, a simulation does not necessarily search for the optimal outcomesin a given situation. Rather, it merely portrays the characteristics of the subject system under varying conditions. This trait implies a requirement for user interaction and choice to select a decision path. A flight simulator, which is constantly requiring user responses to situations, exemplifies this feature.

Simulations are employed when routine management tools, such as linear programming and the transportation method, cannot capture the essential features of a situation. This simulation use reflects the lack of programmable optimality in the parent system. Also, it might imply that the formulation of an optimal solution is possible, but not

resource efficient. The nature of many simulation problems mirrors interacting and dynamic variables that cannot be accurately estimated by probability. To wit, consider the case of a large, multi-user mainframe computer center. The computer center is faced with irregular batch processing requirements, fluctuating interactive processing demands driven by terminal reliability and user workloads, electronic component mean time to failure distributions, variability in external power supply and turnover in personnel staff. Quite simply, a simulation is better suited of representing this myriad of variable components than an analytical or optimization methodology.

Modern simulation traces its origins to Edwin Link. Link sold his first foul-weather flight simulator to the U. S. Army Air Corps in 1933. Combining some boxes, a few instruments and a stick, this flight simulator employed a primitive motion system. World War II saw a proliferation of Link's basic ideas, as countless pilots trained on Link's 'blue boxes'. Technology has significantly boosted simulation, as modern simulation devices can realistically portray flight in a commercial jumbo jet and the space shuttle. Such technology, an outgrowth of computers, sensual enhancement devices and environmental control systems, has effectively relocated reality into an artificial setting. [Ref. 9: pg. 159]

Product applications of simulation can be generically categorized into the following three areas:

1. Decision Training simulation.
2. Physical Response Oriented simulation.
3. System Observation simulation.

These categories are not mutually exclusive, but rather are synergistic elements that combined produce realism and credibility to the simulation user.

Decision training simulations involve confronting the participant with a discrete quantity of alternatives that are intended to strengthen his judgement. This type of simulation is enjoying widespread application in both public and private sectors. The Standard Embarkation Management System (SEMS) simulator, a computer-based simulation device developed by the U. S. Marine Corps for logistics training, is currently being used at Landing Force Training Command, Atlantic and Landing Force Training Command, Pacific. Users benefit from learning in a real-time environment, as well as from being exposed to computers. The organization benefits in that the teaching syllabus was reducible from four weeks to two weeks due to the simulation, and from the simulation being implementable on existing assets (an IBM Series I minicomputer). DeBord and Siebel describe a similar training simulation and benefit accrual in an unnamed private sector organization [Ref. 10: pg. 42].

Physical response oriented simulations prompt decisions from participants, and evaluate the correctness of supplied responses. The nature of the responses serves to generate immediate changes to the external environment surrounding the simulation. These changes connote success or failure at a task, and range in magnitude from losing an arcade game to being the simulated subject of a confirmed kill from an aggressor's air-to-air missile. A recreationally oriented physical response simulation is called the SR-2. Physically resembling a Volkswagen bus supported by three giant shock absorbers, this product of Doron Precision Systems projects an image of road racing, roller coaster rides and a combat fighter mission in Vietnam to twelve simultaneous riders in a four minute timespan. This simulation is complete with sound and wind effects. Realism and reality are stressed in the Singer Link Flight simulation at Williams Air Force Base, Phoenix, where fighter aircraft pilots leave tandem F-4 cockpits drenched in perspiration. William Turner, president of Singer's Link Flight Simulation division, refers to such participant involvement as 'the pucker factor' [Ref. 9: pg. 159].

System observation simulations revolve around the dynamic and continuous interaction of parent system elements. The simulation's purpose is to produce snapshots of identifiable outcome subsets of the parent system. Normally, this type of simulation is employed where multiple

components interact in irregular manners. It can serve as an economical substitute for laboratory and real world experiments. Currently, the Environmental Protection Agency uses this type of simulation to project the effect of pollutants on air basins. Fortune 500 firms use system observation simulations to forecast economic trends and product lifecycles. This type of forecast cannot be commonly solved using traditional analytical and regression mechanisms.

Simulation is having, and will continue to have, a definitive impact on organizations. This impact produces both positive and negative byproducts. Through careful planning and insightful techniques, organizations can exploit these advantages of simulation and avoid the pitfalls.

From the positive perspective, the benefits of simulation can be enumerated as follows:

1. Simulation provides participants the opportunity for repetitive use. In training environments, whether decision or physical response oriented, repetition tends to foster reinforcement of actions. This reinforcement tends to produce a more effective resource when he/she returns to their actual organizational position.

2. Simulation compresses time. This increases its efficiency as a training resource, as well as exposing participants to conditions that might take several years to encounter in on-the-job conditions.

3. Simulations allow users to experience disaster and success without harming the organization. This can produce an intangible maturity factor in the participant. Additionally, this feature is integral

to the credibility of physical response oriented simulations.

4. Simulation remain somewhat of a novelty item. As such, they generally produce high levels of participant interest. This is a vital link to the successful implementation of a training program.

5. A properly developed simulation causes developers and managers to gain a deep and comprehensive knowledge of the parent system. This knowledge base can foster better decisions concerning the controllable variables in the parent system. Also, it frequently generates enhanced intra-organizational communication.

Simulation is a two-edged sword, as the negative aspects can produce significant organizational obstacles. These pitfalls include:

1. Simulation creation and development is a time and resource intensive process. Although formal research does not support it, it is reasonable to assume that simulation is a contributing factor to the ever-rising software cost curves. Two possible options exist to rectify this problem. With in-house developed simulations, all reasonable attempts to retain and reuse this corporate knowledge should be applied. In that simulation development is often a personnel intensive task, this guideline implies personnel retention policies and incentives. If developed externally, serious consideration to sole source acquisition from the same vendor should be given. (Note: This assumes user satisfaction with the initial simulation.) In either instance, the cost and time of reclimbing the learning curve are avoidable by this strategy.

2. Simulation is an art, not a precise science. It is not an emulation of the parent system, and will not identically replicate all system behavior. The solution to this problem lies in initial system definition. If the parent system can be reduced to a simple model that is solvable with analytical tools, simulation should be avoided. This avoidance will prevent the development of an emulation, and the unneeded expenditure of resources.

3. Simulation is frequently perceived by users as the universal solution to all their problems. This mindset is particularly pervasive in the training and

instruction arenas. Two tactics are available to combat the unrestrained proliferation of simulations. First, prior to entering into the development of a major simulation, a feasibility study and cost/benefit analysis should be conducted. If the simulation is beyond the available resources, or fails to meet organizational payback criteria, it should be abandoned. Secondly, The development of organization-wide simulation libraries and user groups can serve to prevent unnecessary duplication of effort within a concern. Hewlett Packard has been extremely successful with this tactic in the applications programs area, and the concept is readily transferable to simulations.

In order to lend some structure and flow definition to the pseudo-science of simulation, the following Simulation Development Lifecycle was developed by the thesis authors. The origins of this simulation development lifecycle are:

1. Boehm's Waterfall Model of the Software Lifecycle [Ref 11: pg. 35 - 46].
2. The generally accepted principles of simulation and modeling [Ref. 8: p. 22].
3. Brooks's insights concerning Project Management [Ref. 12].

The simulation development lifecycle is a hybrid mixture of these sources.

The waterfall model of the software lifecycle was selected for use as modern simulations are increasingly dependent on computers. As earlier discussed, software is the laggard force in the computer revolution. By default, software becomes the weak link in the simulation chain. Integration of the waterfall model of the software lifecycle into the simulation development lifecycle provides a proven model base for this understrength member of the simulation team.

Incorporation of the generally accepted principles of simulation and modeling serves to uniquely identify this lifecycle model as pertaining to simulation.

Lastly, the inclusion of Brooks' ideas reflects the fact that simulations are frequently the result of project management organizations. The hard learned lessons of Brooks and the OS 360 operating system provide a wealth of practical management thought and consideration to this lifecycle. Summarizing Brook's management insights, the following critical themes are evident [Ref. 11]:

1. As a creative activity with 'tractable medium', computer programming is more subject to optimism than other engineering tasks.
2. A large project can only sustain a 30% per year manpower buildup.
3. If you cannot avoid rescheduling, take the necessary time to do a careful job.
4. When reducing activities, explicitly trim the tasks.
5. Concrete, measurable milestones are necessary in project management.
6. Do not defer action on problems to specified review periods. Take action on problems when they surface, and distinctly separate status review from problem - action efforts.

Ignoring this rich collection of real world experience would be a a shortsighted folly, and tantamount to academic heresy.

The simulation development lifecycle is an eight step, iterative concept. It is composed of the following phases:

1. The Simulation Feasibility phase.

2. The Simulation Requirements/Specification phase.
3. The Conceptual Simulation Design phase.
4. The Detailed Simulation Design phase.
5. The Simulation Module Creation phase.
6. The Simulation Module Integration phase.
7. The Simulation Implementation phase.
8. The Simulation Maintenance phase.

These phases represent a logical and systematic approach to simulation. They are interdependent, but not necessarily sequential. Sequential progress is an ideal and textbook path through the lifecycle. The majority of simulations generate feedback, which may dictate backstepping to review and revise a previously completed phase. Enveloping the entire lifecycle is the continuous verification and validation cycle. Verification is concerned with the relationship between the developing simulation and the simulation specification. Validation refers to the applicability and value of the simulation to the actual end user. This verification and validation cycle permeates all aspects of the lifecycle, ensuring the simulation is being constructed correctly, and that the correct simulation is being produced.

The simulation feasibility phase includes the macro definition of the parent system, the performance of a cost/benefit analysis, answering of the rhetorical analytical/optimization method versus simulation question,

and a schedule analysis and preliminary development for the projected simulation project. Several GO - NO GO decision points are inherent in this phase. Also, the core members of the simulation development project should be assembled into a team during this phase. Neglecting to invest adequate resources (manpower, financial and time) in this phase can build a weak foundation for the entire simulation. Systems analysis techniques can provide useful tools to the simulation team throughout the feasibility phase. They offer the flexibility to recognize technology and financial considerations, as well as handling much of the conceptual thought present in this phase.

After a solid 'GO' decision on the simulation has been reached, the simulation requirements/specifications phase commences. Included herein is the parent system micro definition statement. This should encompass the limits and parameters, functions, purpose and effectiveness indicators to be drawn from the parent system. All these qualities should be testable. This testability is critical to the validation/verification cycle. The simulation support environment must be comprehensively defined, and should include the support functions, interfaces, facilities and support performance measures. This is also the time to establish concrete, measurable simulation project milestones. Finally, the entire simulation project team should be assembled. Familarization training should follow.

Formation of a firm simulation project team in this phase may raise startup costs. However, personnel stabilization and solid training provide protection against the occurrence of Brooks's Law [Ref 11: pg. 25]. Throughout this phase, heavy end user involvement is integral to the production of verified and validated requirements and specifications.

Proceeding into the conceptual simulation design phase, the simulation project team transforms and abstracts the parent system into models. The previously defined requirements and specifications provide the direction and guidance for this decomposition. Flow diagrams, in a standardized format, can serve as meaningful communication tools and form a documentation foundation. If the simulation sponsors desire a prototype, it should be accomplished in this phase. The definitive statement of support environment and simulation relationships is completed in this phase. Draft simulation users manuals, and test plans are assembled. This phase serves to bridge all preceding efforts with the remaining phases of the project. As such, a careful review of problem areas, with time parametered corrective action, is appropriate.

Logically stepping into the detailed simulation design phase, the simulation moves from a conceptual state to a carefully planned and structured development sequence. Micro support environment decisions are made. Precise definition of previously grey areas is finalized.

Documentation continues to grow, and should portray a clear simulation decision chronology to date. Models are grouped into macro modules. All tasks short of physically integrating the support environment with the simulation are completed.

Entering the simulation module creation phase, the physical integration of the simulation and the support environment occurs. In a computer-based simulation, this is the translation of the macro modules into complete, documented computer programs. Regardless of the underlying support environment, the outcomes of this phase are the simulation modules. These modules must be tested on a unit basis. The previously prepared test plans, reflecting the requirements and specifications, serve as the performance barometer for the modules. End user involvement in the unit testing reinforces the validation and verification phase.

Upon satisfactory completion of module testing, the individual modules are assembled into a complete simulation system in the simulation module integration phase. Module interface is a critical consideration in this phase. Likewise, documentation must be updated to reflect the module integration and related decisions.

The simulation implementation phase involves placing the fully functional simulation system in the hands of the end user. Inclusive in this process are on-site systems testing, user training and user feedback. This phase is the

acid test for the effectiveness of previous validation and verification cycles. While this step may appear to be anticlimatic after the previous efforts, it remains integral to ensure that the users receive a debugged simulation, understand the operation of the simulation and are adequately trained in their role as simulation participants.

The capstone to this lifecycle is the simulation maintenance phase. Commencing after the simulation is 100% fielded, it covers the gamut from correction of simple logic errors to complete revision and/or replacement of the simulation system. Historically, this phase is the most resource hungry portion of the lifecycle. Simulation maintenance should be carefully accomplished in response to user requirements, take into account support environment constraints, and should parallel parent system evolution.

To summarize, the simulation development lifecycle is a cradle to grave management plan for simulation systems. It intends to interject structure and direction to a somewhat fluid and wandering process. This type of organization and composition is purposely aimed at reducing the inherent risk present in this type of multi-dimensional project maangement effort. While it is not a guaranteed recipe for simulation success, it provides an organization with a solid framework to mold and craft to each individual application.

III. PROBLEM DEFINITION AND INITIAL DECISION PHASE

A. REVIEW

From the previous chapter, two central themes continually surface and merit restatement:

1. A drive by managers to understand the meaning and utility of the AIS, as well as harness its power, is frequently being thwarted by the saturation of mainframe hardware resources and the lack of adequate hardware and software tools to meet the needs of decision makers.

2. Simultaneously, many managers are seeking to exploit the capabilities of the microcomputer to enhance their decision making function.

The obvious intersection of these two tendencies is the simulation of an AIS on a microcomputer.

B. PROBLEM STATEMENT

This intersection is the basic hypothesis to be pursued in this thesis. Specifically, this thesis will address the problem of using a microcomputer and related software to simulate a mainframe AIS in a feasible and realistic manner. To probe this problem, the following sequential approach will be used:

1. Software Determination: Through the use of a decision model incorporating user needs and requirements, software selection will be accomplished.

2. Hardware Determination: Using another decision model reflecting user needs and requirements, the hardware selection will be made.

3. The Simulation Development: Through application of the waterfall model of the software lifecycle, the simulation will be designed, coded and implemented.

4. Test and Evaluation of the Simulation: A formal validation and verification phase, which encompasses the entire project lifecycle, will be executed to ensure the 'rightness' and correctness of the simulation.

During the course of this simulation development and application, several research questions will be investigated. Specifically, answers to the following will be sought:

1. Are microcomputers technically capable of executing software packages that simulate a major AIS?

2. What are the positive and negative outcomes associated with simulating a mainframe AIS on a microcomputer?

3. What are the essential features that must be present in an AIS simulation to ensure that it is a representative model?

4. How can an AIS simulation be designed and developed to maximize user satisfaction within organizational and technological constraints?

5. Are microcomputer-based AIS simulations a viable option for future applications?

The respective findings will be presented in the thesis conclusion.

The focal point of the problem statement revolves around the term 'simulation'. Recalling from the previous chapter that a primary objective of simulation is to garner an understanding of the system and its related behavior, this thesis will emphasize this goal. This emphasis is driven by the resource sponsor of the project, the Fiscal Director of

the Marine Corps, and the intended application of the resultant product as a training package for the Marine Corps Practical Comptrollership Course (MCPCC). Further discussion of this matter is presented later in this chapter.

The inherent mechanism of any simulation is the model. The model is merely a representation and abstraction of certain features of the parent system. The key fact is that the model is designed to abstract and represent only certain features of the parent system. This is a point of differentiation for simulations, as a simulation tends to imitate and not directly emulate or replicate the exact behavior of the parent system. The extent and nature of this particular imitation will be explicitly addressed in the system definition phase of the simulation development lifecycle.

As with any science, principles of simulation exist to provide humans with a solid, directional framework. This simulation will be guided by these principles, as follows: [Ref 8: p. 4]

1. Principle: The simulation must be easily understood by the ultimate end user. Application of this principle will be via the following methods:

    a. Explicit designation of a user group.

    b. Continuous and iterative communication and feedback between the user group and the system developers during all phases of the simulation development lifecycle.

c. Development and implementation of a formal feedback loop after fielding of the simulation.

2. Principle: The simulation must be goal or objective directed. This principle will be applied with the following methods:

   a. Development of a software decision model enveloping the user needs and requirements.

   b. Development of a hardware decision model incorporating user needs and requirements.

   c. Specific user designation of simulation goals and objectives, commencing with specification of the system elements and components to be simulated.

3. Principle: The simulation must be robust. Principle application will be through the following methods:

   a. Incorporation of error modules into the product and detailed design phases of the simulation development lifecycle.

   b. Extensive simulation testing by representative users prior to operational fielding.

   c. Integration of hardware characteristics into the man-machine interface of the simulation.

4. Principle: The simulation must be complete with regard to the critical issues of the parent system. This principle will be realized with the following methods:

   a. Front end loading of user/developer effort and communication in the system definition phase of the simulation development lifecycle.

   b. Integration of the critical issues of the parent system into the hardware and software decision models.

In reviewing the above principles and the means to satisfy them, the importance of communication becomes self evident. This communication took the form of weekly meetings between

the user group and the system developers between April, 1983 - December, 1983. Heavy initial emphasis and discussion was invested in the system definition phase, consuming approximately 2 1/2 months of this time period. This effort was complimented by selected user group testing of robustness of the system on a module-by-module basis. Lastly, the communication between the users and developers was two-way, iterative and not limited to the formal weekly meetings.

The point to be gleaned from the preceding discussion of simulation principles is that clear and continuous communication is the backbone of the simulation development lifecycle. It provides the direction and planning necessary to deliver a verified and validated simulation. The absence of such communication is a certain indicator of incumbent failure.

C. DETERMINATION OF THE AIS TO BE SIMULATED

The first requisite decision in this thesis is the determination of which specific AIS to simulate. This is a baseline decision, as it serves to direct all future efforts.

As with most decisions, the AIS simulation determination decision is a multi-faceted one. The actual decision process proved to be too intangible to quantify into a

decision model. However, the following decision factors were evident:

1. The AIS would be selected from a public sector application. Rationale for this choice are:

   a. Many public sector organizations have a far richer background in AIS operations than private sector firms.

   b. Access to AIS information in public sector organizations is generally easier than in the private sector. This access to information, ranging from source code to software design decisions, is critical throughout the simulation development lifecycle.

   c. Both thesis authors have had moderate exposure to public sector AIS's. This factor eliminates some of the AIS relearning usually required in the simulation development lifecycle.

2. The AIS will interactively communicate with the user. Supporting rationale for this requirement are:

   a. Interactive communications intensifies the feedback loops between the user and developer. This is viewed as having a positive effect upon the simulation development lifecycle.

   b. Increasingly, computer-based simulations are entirely screen oriented. This screen orientation dovetails nicely with an interactive AIS.

   c. The entire realm of computing is evolving from batch to interactive processing. Selection of a batch oriented AIS to simulate might prematurely destroy any practical applications of the simulation.

3. The simulation developers must have complete access to the entire parent AIS package. Supporting rationale for this factor are:

   a. The rationale ensures the presence of adequate information to allow simulation and to prevent emulation/replication of the parent AIS.

b. Such access is integral to the incorporation of abstraction and information hiding in the simulation development lifecycle.

c. This requirement ensures the simulation developers will be kept abreast of any changes to the parent AIS during the simulation development lifecycle.

4. The AIS must be DBMS oriented. Supporting rationale for this factor are:

   a. As earlier noted, the AIS trend is drifting in the conceptual direction of database management.

   b. Database orientation compliments screen driven, interactive user communication.

5. The AIS simulation must ultimately be applicable in a non-academic environment. Supporting rationale are:

   a. This consideration is critical to gaining a project sponsor. Sponsorship is required to provide the financial resources to procure the software and hardware assets.

   b. This linkage is personally important to the thesis authors, due to their previous military backgrounds.

6. The AIS simulation decision will be a language and software independent decision. Supporting rationale for this factor are:

   a. This type of decision independence reinforces the research aspects of this project and the future applicability of results.

   b. Failure to recognize this point might tend to unnecessarily delimit the scope of this thesis.

These six considerations provided an analytical framework to screen AIS simulation candidates. This analysis took approximately one (1) week. It produced three primary candidates:

1. The Marine Standard Supply System (M3S): This is the Marine Corps' new supply system for the operating and support establishments. It is database oriented, and intended to provide a uniform supply system to all Marine Corps users. It is interactively driven, and is designed to replace the current family of 2nd generation supply software with 4th generation software products in the Marine Corps.

2. The Standard Accounting and Budget Reporting System (SABRS): SABRS is the financial compliment to M3S. Again, it is database oriented and applicable in the operating and support establishment of the Marine Corps. Fourth generation software will drive SABRS.

3. The PRIME Enhancement Project (PEP): PEP is a layered on enhancement to the PRIority Management Effort (PRIME), the current financial management system used in the Marine Corps support establishments. It is designed to transform PRIME from a batch system to a user-perceived interactive system through the use of the ADABAS database management system.

Closer analysis of these three candidates led to the selection of the PRIME Enhancement Project as the specific AIS to simulate. The decision to simulate PEP is based upon the following:

1. M3S and SABRS are in the early development stages. They contain strong concepts which have yet to materialize on a CRT. This level of development makes them too immature to simulate. PEP is currently operational at several Marine Corps sites.

2. A project sponsor, the Fiscal Director of the Marine Corps, has a high level of interest in both microcomputers and the results of this simulation.

3. No restrictions on access to PEP code or software development documents exist for purposes of this thesis.

4. Doctrinally, PEP is the bridge between many of the current Marine Corps' batch processing AIS's and the planned, interactive AIS packages. Doctrine application includes the integration of PEP into SABRS as a primary module.

5. The PEP Development team, composed of Major R. Cowen, USMC, Captain R. Robinson, USMC and Ms. B. Hille, are strongly supportive of this effort. This type of organizational backing is viewed as key to overcoming potential obstacles in the simulation development lifecycle.

Before proceeding, a further description of PEP is required. PEP is an in-house development of the Commandant of the Marine Corps (Code FD) and the Central Design and Processing Activity, Marine Corps Development and Education Command (MCDEC), Quantico, Virginia. It is a financial manager's system designed to permit interactive transaction input to a batch update file with full editing, allow the user to view information in certain PRIME master files, and to permit supervisors to review and delete transactions, maintain table files and extract transactions. PEP is currently operational at MCDEC and Marine Corps Logistics Base, Atlantic, Albany, Georgia. Application is planned for the remaining Marine Corps support establishment.

From a software perspective, PEP is a layered on package to PRIME. PRIME is a second generation, COBOL based, batch oriented financial management system. PEP's source code is written in NATURAL, the command language of ADABAS. ADABAS is a network architecture database management system. PEP couples in-house written NATURAL programs and ADABAS functions/procedures to perform user-designated tasks. Historically, it is a logical step forward for Marine Corps software from the era of batch-driven, MARK IV inquiries.

Hardware-wise, PEP is implemented on Amdahl 470/V7 mainframe computers. It is oriented to the 24 line X 80 column format and keyboard of an IBM 3270 terminal. PEP is supported by the OS 370 operating system.

D. THE USER GROUP

The next logical step in this project involved the formation of a user group. User group formation began with the definition of the actual users of PEP, and with the definition of the users of the PEP simulation. This duality of definition was designed to balance the needs of the real users of PEP with the possible constraints facing the users of the PEP simulation. Following the user group definition, formal designation of the user group members would ensue.

The actual users of PEP are the supporting establishment comptrollers and their staffs throughout the Marine Corps. This is a diverse group that contains few demographic commonalities. They are both military and civil service employees. Educational levels vary from high school to post master's degree studies. Age varies from 18 to beyond 50. Their work experience is likewise varied, ranging from entry level positions to the top financial management billets in their commands. Further isolation of common demographics for this group failed to produce any usable points.

The PEP simulation users fell into a similar category. The project sponsor desired that the PEP simulation be

integrated into the curriculum of the Marine Corps Practical Comptrollership Course (MCPCC). This course is conducted twice annually at the Naval Postgraduate School (NPS). Students at the MCPCC come from financial activities throughout the Marine Corps. Their demographics are somewhat more identifiable:

1. Rank/rating.
    a. Military students: An officer, WO-1 and above.
    b. Civil Service students: GS-8 and above.
2. The student's current billet must require or be related to financial management.
3. Student nomination is performed by the local command. Student selection is performed by the Fiscal Director of the Marine Corps.

These demographics provided a somewhat tighter definition of who should compose the user group. However, practical considerations as to the availability of such personnel for active participation in a user group, and the time constraints on system development prohibited the integration of actual or perspective students into the user group. These same considerations prohibited the integration of actual PEP users into the user group.

Given this limit, it became necessary to examine local resources. This proved to be a rich source of users, as numerous Marine Corps students at NPS have financial management backgrounds. From this student inventory, the following user group emerged:

1. Major R. H. Myers, USMC: Major Myers was a student in the financial management curriculum. His military background included a varied supply and fiscal billets. He would serve as the head of the user group.

2. Major D. Grimm, USMC: Major Grimm was also a student in the financial management curriculum. His background included experience at the Fleet Marine Force level of financial management.

3. Captain S. G. Shumway, USMC: Captain Shumway, a financial management student, brought a rich variety of operational and supporting establishment financial management to the user group.

These three financial managers composed the formal user group. The diversity and scope of their combined backgrounds provided the depth necessary to accurately represent the actual users of PEP and the PEP simulation users.

Thus, the user and development groups were formally designated. Additionally, Lieutenant Colonel J. F. Mullane, Jr., USMC became the defacto project manager of this effort. LtCol Mullane, the NPS Marine Corps Representative and a financial management instructor at NPS, provided the direction and guidance to mold the user and development groups into a formative project team. Also, he served as the formal interface with the project sponsor.

This chapter has provided a broad overview to the components of the problem. Also, it has outlined the methodology and principal decisions made prior to the actual problem resolution. The remaining chapters focus on

specific portions of this problem resolution, and are intended to highlight the types of decisions and tradeoffs inherent in this problem resolution.

IV. SOFTWARE

A. OVERVIEW

The use of a microcomputer to provide a simulation support environment involves two integral and related components: hardware and software. This chapter will address the software issue. Hardware will be covered in chapter V. The software topic will be introduced in this chapter with an initial definition of the software requirements. This sector addresses the semantics of software requirements, the relationship of software requirements to the simulation development lifecycle, and the requirements rationale. The managerial source of the software requirements will also be identified. After the requirements definition will be a presentation of the software decision model. Included in this presentation will be the model's purpose, it's structure, and a brief discussion of the modeling support environment. After the the software decision model has been developed, four alternative software solutions will be introduced. Definitions, applications and examples, and the respective advantages and disadvantages of each software alternative will be detailed. All of the software alternatives will be evaluated with the software decision model. The objective of this process is to select the most desirable software

alternative. The chapter conclusion will be an in-depth review of the specific software product selected for use in the simulation and a concise description of the software acquisition process.

B. DEFINITION OF SOFTWARE REQUIREMENTS

When a consumer has a need, he visualizes a product to satisfy the need. He refines the thought by clarifying the exact nature of his need, and developing evaluation criteria to judge alternatives that can satiate the need. This common need determination process, a sequence every shopper has gone through countless times, is analogous to the definition of software requirements in the simulation development lifecycle. A need is known to exist (software), and the concerned parties consciously consider how to best gratify this need through the definition of requirements.

Moving from this conceptual perspective to more concrete ground: the goal of requirements definition is to create a complete, consistent and unambiguous specification of WHAT the software product will contribute to the simulation. This is an intentional emphasis of the WHAT, as the corresponding HOW is the responsibility of the design phase [Ref. 13: pg.1]. Within the software requirements definition procedure, two terms are prevalent: requirements and specifications. Requirements tend to deal with the identification of objectives and the understanding of the

problem. They are macro in scope. Specifications are the detailed description of what the software must be, in terms of functions and interfaces. Specifications tend to be more limited in scope than requirements. The relationship between requirements and specifications is comparable to the correspondence between strategic and operational management. Requirements describe the big picture, where as specifications portray the situation from a more detailed level. This relationship forms a natural hierarchy. Violation of this hierarchy can produce upheaval throughout the remaining phases of the lifecycle.

The software requirements definition phase has a unique and vital relationship to the simulation development lifecycle. This relationship is a direct function of the following:

1. As previously noted, software is usually the most vulnerable component of a computer system. As such, it represents the highest risk to any proposed project. This intensifies the need for a clearcut statement of software needs, relative to the proposed system. Lack of proper definition introduces further uncertainty to the software, and can produce a destructive ripple effect throughout the lifecycle. To minimize this risk, software is routinely the first system component to be defined after the feasibility analysis. This approach appreciates the highest risks up front, and defers the more stable system components to a later decision point.

2. The software requirements definition provide s a precise and clear statement of the software system. This generates a baseline for the validation and verification cycles. Also, it provides a foundation for test plans and test procedures.

3. Software requirements concretely define the 'What' of the software. This is a requisite, sequential step that appropriately precedes the 'How' of the design phases.

4. Lastly, the requirements definition is the user's explicit message to the developers. This must be a distinct and unclouded communique to ensure the users receive the desired end product.

Within the federal government, requirements are further categorized as mandatory and desirable. Mandatory requirements represent absolute needs that must be satisfied in the delivered product. Failure to fulfill a mandatory requirement eliminates an alternative from further consideration. Desirable requirements are features that a user would prefer to have, but is not willing to apply as an elimination factor between alternatives. Generally, desirables are enhancement features to the basic product.

Software requirements definitions flow primarily from the users, and supplemented by the developers. This joint input to requirements tends to blend the user's desires and perceptions with the technical realities that the developers must contend with. In the PEP microcomputer simulation, the previously discussed user group and development team served as the source of these inputs.

Requirements definitions can assume three administrative configurations: Informal, Formatted or Formal. The PEP microcomputer simulation used the informal form. It was deemed appropriate for the following reasons:

1. The degree of financial risk inherent in this endeavor was insufficient to justify formatted or formal methods.

2. With the presence of the parent system and the supporting documentation, there was insufficient technical risk to use formatted or formal configurations.

Using the informal requirements definition configuration, the PEP simulation software requirements were further categorized into general software requirements, software engineering related requirements, and database handling requirements. Each individual requirement was defined in plain language. The supporting rationale, type designation (mandatory or desirable), and the requirement source were appended to the definitions.

Starting with the general software requirements, the following separate individual requirement criteria were developed:

1. Requirement: The selected software must permit development of the microcomputer simulation in a 3 to 4 month time period.

    a. Rationale: The lack of development team resources (two designers/programmers) drove this requirement. The December MCPCC class implementation date defined the 3 to 4 month time period. Finally, this requirement was chosen to emulate the time compression present in many project management environments.

    b. Type: Mandatory.

    c. Source: Joint input by the user group and the development team.

2. Requirement: The software must be fully transportable between microcomputers.

a. Rationale: This requirement reflects the desire to potentially apply the product outside of the thesis environment. Also, it reflects the current lack of formally accepted microcomputer standards within the Marine Corps.

b. Type: Mandatory.

c. Source: Joint input by the user group and the development team.

3. Requirement: The software package must support documentation and data dictionary production.

a. Rationale: This requirement is derived from the the principle that 'Documentation is the Software Product' in systems analysis [Ref. 14: pp. 24]. Also, it was included to support the inevitable future modifications to the simulation.

b. Type: Mandatory.

c. Source: The development team.

4. Requirement: The software must permit expansion and contraction of the simulation.

a. Rationale: As PEP is designed as a primary module of SABRS, this requirement is intended to facilitate the eventual simulation of SABRS from this foundation. Also, this requirement was included to allow simulation upgrades to parallel any upgrades in the parent system.

b. Type: Mandatory.

c. Source: Joint input from the user group and the development team.

5. Requirement: The software must support interactive programming.

a. Rationale: Interactive processing is the fundamental element of the parent system. As such, it must be present in the simulation.

b. Type: Mandatory.

c. Source: The user group.

6. Requirement: The software must provide time responsive execution on a conventional technology microcomputer.

   a. Rationale: The time responsiveness was deemed critical to holding the student's attention during anticipated system use. Also, time responsiveness was necessary to maintain a resemblance to mainframe processor response times.

   b. Type: Mandatory.

   c. Source: The user group.

7. Requirement: The software must support real time simulation of PEP in a stand alone environment.

   a. Rationale: This need was intended to minimize the complexity of system use for students. Student use was projected to be in hotel rooms or government quarters, where peripheral support would not be readily available.

   b. Type: Mandatory.

   c. Source: The user group.

8. Requirement: The software must support numeric accuracy for mantissa up to and including 1 * 10**8 with with two decimal place formats.

   a. Rationale: This numeric accuracy is an essential element of the parent system to simulate.

   b. Type: Mandatory.

   c. Source: The user group.

9. Requirement: The software must facilitate the integration of security features e.g. passwords, unit designations from the parent system into the simulation.

   a. Rationale: This was deemed an essential feature of the parent system to simulate. The intention is to familiarize the students with the security features of computers, and to foster thought on computer security.

b. Type: Mandatory.

c. Source: The user group.

10. Requirement: The software should be primarily oriented towards non-tactical data processing.

    a. Rationale: This requirement was adopted to prevent the use of embedded software packages. Embedded packages tend to imply the presence of functions i.e. device sensing, absolute reliability through software layering that are not critical to the simulation. Also, the embedded packages tend to tax the resource efficiency of conventional microcomputers due to the software redundancy.

    b. Type: Mandatory.

    c. Source: The development team.

The next requirements category focused on software engineering considerations. This step was a logical progression from the general software requirements. The resulting requirements packages were:

1. Requirement: The software package must support structured programming.

    a. Rationale: Inclusion of this need was designed to increase development team efficiency, permit traceability of flow in programs, and enhance the readability of the source code.

    b. Type: Mandatory.

    c. Source: The development team.

2. Requirement: The software package must support separate_modular development of the simulation.

    a. Rationale: This trait is desired to allow the implementation of discrete unit testing, permit a logical and learning curve phased development sequence, enable the simulation to embrace the reusable code concept, and permit the application of program stubs. Additionally,

modular development is the backbone of the conceptual design phase to the module integration phase of the simulation development lifecycle.

b. Type: Mandatory.

c. Source: The development team.

3. Requirement: The software package must allow the implementation of information hiding.

a. Rationale: Information hiding, the abstraction of common system functions and the subsequent access constraints to the 'How' of the function, is critical to the separation of the design and module creation phases of the simulation development lifecycle. It permits developers to encounter complexity in understandable doses. Also, it will cause the simulation to present a realistic exterior to users and suppress the internal details from the user's view.

b. Type: Mandatory.

c. Source: The development team.

4. Requirement: The software must permit separate compilation/interpretation of modules and subprograms.

a. Rationale: This function is required to permit the timely completion of the module creation and module integration phases of the simulation development lifecycle. Additionally, it is needed to support top-down testing.

b. Type: Mandatory.

c. Source: The development team.

5. Requirement: The software package must possess a comprehensive library of built-in functions, and permit the development of user defined procedures.

a. Rationale: This requirement is needed to allow system development within the defined time period.

b. Type: Mandatory.

c. Source: The development team.

6. Requirement: The software package must display understandable syntax errors, and contain logical error detection functions.

    a. Rationale: The presence of these characteristics is required to support the programming effort. Additionally, they will tend to aid in isolating logic errors.

    b. Type: Mandatory.

    c. Source: The development team.

The next component of requirements definition dealt with database handling requirements for the software. The following requirements were defined from the database perspective:

1. Requirement: The software package must support numeric (integer and real), character and boolean data types.

    a. Rationale: The presence of these three data types is essential to the simulation of the parent system. Also, the presence of boolean data types is intended to support program decision branching.

    b. Type: Mandatory.

    c. Source: The development team.

2. Requirement: The software package must facilitate menu driven query of the database.

    a. Rationale: This subset of the interactive processing requirement is needed to simulate the Inquiry module of the parent system.

    b. Type: Mandatory.

    c. Source: The user group.

3. Requirement: The software package must permit real time update of databases.

    a. Rationale: This requirement was included to allow for the time compression of the pseudo-

interactive processing of the parent system. It reflects a conscious tradeoff decision to depart from the actual characteristics of PEP in order to retain the student's interest in a microcomputer support environment.

b. Type: Mandatory.

c. Source: Joint input from the user group and development team.

4. Requirement: The software package will project a logical view of the data, while supressing the actual physical data structure and linkages.

a. Rationale: This requirement incorporates the principle of abstraction and information hiding that are present in the parent system. Also, it recognizes the microcomputer knowledge level of the average student user and the potential for confusion with trivial details from the user's perspective.

b. Type: Mandatory.

c. Source: Joint user group and development team input.

Collectively, these three categories of requirements define 23 distinct functional software specifications. Despite this broad base, three commonly included considerations were excluded: networking, database integrity, and the structure of the database in the parent system.

Networking was omitted because of the requirement for the simulation to operate as a stand alone system. As such, there was no forecasted need for networks in the intended application.

Database integrity was not considered, as the simulation is by hardware definition a single user system. Additionally, it was envisioned that any potential database integrity problems could be eliminated by invoking a file initialization routine at the beginning of each user training session.

Lastly, the structure (both physical and logical) of the database in the parent system did not receive any attention. Incorporation of this facet as a requirement would drive this project in the direction of emulation, not simulation. Also, it would tend to violate the abstraction principle requirement. Finally, defining this as a requirement is really a statement of 'How' the simulation should operate. This disregards the 'What' concept of requirements, and could serve to artificially limit the range of software alternatives.

C. THE SOFTWARE DECISION MODEL

A multi-dimensional decision mechanism was considered the best vehicle for integrating the 23 software requirements into a format that supported candidate software product evaluation. A quantifiable decision model was constructed using this mechanism. The objective of this decision model was to quantitatively evaluate various software alternatives. The previously defined software requirements formed the basis of this evaluation process.

The structure of the software decision model is comprised of the decision criteria and the variables. The decision criteria represented the foundation of the model. In this instance, the previously defined software requirements constituted the decision criteria. This yielded 23 separate evaluation criteria for the decision model. Variables were used to assign a relative value to the respective alternatives for each decision criteria. In the software decision model, each alternative would be assigned a variable corresponding to individual requirements in the 0 to 100 range. A value of 0 would denote the complete failure of a software alternative to fulfill a requirement. Conversely, a value of 100 would represent a complete compliance with a requirement. Values between 0 and 100 represent a partial requirement fulfillment. The assignment of these values, to the software alternatives, was the responsibility of the development team. The development team used the generally accepted characteristics of each particular software alternative as a guide in awarding these values. Further discussion of these generally accepted principles is contained later in this chapter. Upon value assignment for each decision criteria to all the alternatives, the values were summed by alternative to determine the total requirement value for each software alternative. Then, the total requirement

values of each alternative was divided by 23. The result of this division was the mean software requirement value for the alternative. The software alternative with the highest mean software requirements value was the most likely candidate to be selected as the software component to utilize in the simulation, given no unforeseen complications or circumstances.

This decision model has several strengths. Foremost, it directly incorporates the software requirements. Additionally, it is a fairly simple, two step solution to a multi-faceted decision. Lastly, it has the inherent flexibility to recognize the strengths and weaknesses of each alternative.

Conversely, a subjective weakness is present in this deterministic model. It relies on the development team to assign variable values. Such reliance could diminish objectivity and introduce bias to the decision. This potential drawback is present in any qualitative measurement model. The following factors serve to offset this possible pitfall:

1. The education and experience levels of the development team.
2. The alternative with the highest mean software requirements value will not be blindly procured. Rather, discussion with the project manager and user group will ensue. This review process is designed to test the validity of the model, and identify any subjectivity or bias.

The software decision model was constructed in the Interactive Financial Planning System (IFPS) modeling language by Execucom. This support environment was selected because of its availability at NPS, for its ease of use and out of respect for the strong allegiance IFPS has in the corporate world. The IBM 3033AP computer, also located at NPS, provided the necessary hardware support.

D. THE SOFTWARE ALTERNATIVES

The software alternatives were chosen from a generic perspective. Further, they reflect the conventional microcomputer technology that is present in the hardware candidates. For purposes of this thesis, the term 'conventional microcomputer technology' is definable by the presence of the following:

1. Primary Microprocessor: A member of the Z-80 family, an 8088, 6502 or 80186.
2. Primary Operating System: Apple DOS, C/PM 80, C/PM 86, or MS DOS.
3. RAM: 64 to 712KB.

This definition is a function of the hardware requirements presented in chapter V, and the fact that this thesis has no requirement to push the leading edge of hardware technology to satisfy the project requirements.

The conventional microcomputer technology constraint, coupled with other factors, eliminated the following alternatives from further consideration:

1. Artificial Intelligence languages (LISP, Prolog): While interpreters and compilers are available, these software packages lack the internal library of functions needed to permit the development within the stated timeframe. Also, these packages routinely require RAM in excess of conventional microcomputer configurations.

2. Mainframe DBMS software (SEQUEL, ADABAS): While these alternatives would have significantly simplified the development phase, there are currently no commonly marketed compilers or interpreters for these languages in the microcomputer realm.

3. ADA: The lack of a certified, operational DOD compiler for ADA, coupled with its integral relationship with embedded systems, provided the reasoning to not consider ADA as a software alternative. Additionally, there are only non American National Standards Institute (ANSI) subsets of ADA, such as JANUS, that are currently available for microcomputers.

This elimination process produced the following four software alternatives for evaluation:

1. Assembly Language.

2. High Order Language.

3. Microcomputer DBMS Command Languages.

4. Simulation Languages.

Assembly language is a low level, processor specific language. It relies on mnemonic codes to structure the language. These mnemonics have a one-to-one relationship with the machine language instructions that operate the CPU. This one-to-one translation produces two significant effects. First, it results in much more efficient code than most compilers can produce. It also gives the

programmer direct control over the bare machine via the basic microprocessor instruction set. Most high level languages do not provide an interface with the hardware at such a low level. These positive points create a second, disadvantageous result. A single line of high level code can invoke operations that might require anywhere from 8 to 200 lines of assembly code. This type of relationship makes assembly language coding a labor intensive, error prone process. This negative condition has been partially muted by the development of macro instructions and subroutines. Macro instructions are single instructions which invoke a sequence of machine language instructions when executed. Subroutines are the application of the reusable code concept to assembly language. They have proven to be an extremely powerful tool when made available to programmers via a library. The library eliminates 'reinventing the wheel' when a common function i.e. sorting, square roots is needed. Another drawback to assembly language is the amount of complexity present. It requires a one-to-one interface with physical devices, and presents the programmer with a substantial mnemonics dictionary to grasp and apply.

Despite this mixed review, assembly language remains a significant factor in the microcomputer market. Traditionally, microcomputers have had limited memory and processing resources. This physical constraint favors an

efficient coding method like assembly language. Complimenting this efficiency is the increased device control achievable with assembly language. This enhanced device control can result in 'user friendly' interfaces with operators. The applications spectrum for assembly language covers the entire software gamut. Two prominent products, Ashton Tate's dBASE II and MicroPro's WordStar, demonstrate this variety. dBASE II is a relational database management system coded in assembly language. WordStar is an extremely popular word processing program that is assembly language based. Additionally, literally hundreds of system software applications, ranging from single task utilities to complete operating systems, are source coded in assembly language.

High order languages (HOL) are the backbone of traditional data processing. Structured with language specific syntax and operator definable semantics, HOL's are converted into machine language by a compiler or interpreter. The compiler or interpreter effectively transforms HOL statements into machine language macro instructions. HOL's have continually attempted to permit programming with English like words. HOL's are considered to be a fairly machine independent language at the applications program level. The dependency comes at the compiler level, which is routinely linked to a family of processors.

HOL's produce a windfall of advantages. First, they are much easier to comprehend than their assembly language counterparts. Frequently, a HOL will contain a broad library of functions and will support the execution of user defined procedures. HOL source code is more readable than assembly language source code. HOLs are more forgiving in coding than assembly language, and offer strong diagnostic capabilities to simplify program debugging. The net result of all these benefits is that HOLs provide a less labor intensive program development environment.

HOLs are not without pitfalls. Despite several significant attempts, HOLs tend to reflect a clear and distinct orientation. This orientation is a function of programming language design decisions towards a given type of application made by the language's authors. For example, COBOL is particularly well suited for file processing. However, it is inefficient when used for scientific applications. The reverse tends to be true for FORTRAN, which is scientifically slanted and inappropriate for file processing. APL is mathematically directed, as is ALGOL. Pascal originated as an educational language designed to emphasize structured programming and top-down design, and has evolved into several business application areas. The outcome of these examples and this orientation bias is that HOLs, when viewed individually, lack a true, general purpose

applications capability. PL/1, developed by IBM, is the best attempt to date to create a flexible, multi-purpose HOL.

The presence of a compiler and applications programs create increased memory and processor requirements for HOLs. Likewise, many microcomputer HOL compilers are not optimized with regard to HOL - machine language transformation. These two factors tend to cause HOLs to push the resource limits of microcomputers.

Despite this, microcomputer HOL applications have evolved and will continue to flourish. COBOL, FORTRAN, and Pascal microcomputer compilers are commonly available. Digital Research's PL/1 compiler is an industry innovator, as it produces better machine language optimization than most of its mainframe counterparts. An entire industry has developed around microcomputer games, many of which are written in BASIC. Lastly, applications are frequently written in HOL for microcomputers. These applications are then compiled to produce a machine language program, commonly identified by the .COM filetype, and used in their machine language version. This approach realizes the flexibility of an HOL in design and coding, while avoiding the need for a compiler and compile time at execution.

Microcomputer DBMS command languages are applications development languages. Logically layered above the database

in the software hierarchy, such languages tend to explcit the query functions of the database manager. This approach allows for far reaching manipulation of the database. Coupled with this effect is the fact that these languages are relatively less restrictive in standards and semantical structure than the previously reviewed software alternatives. The result of this coupling is a powerful and imaginative tool that allows a user to maximize the utility of the database concept.

Normally, a DBMS command language permits the user to define procedures, and supports structured programming and top-down design. Such languages normally use a DBMS compiler or interpreter to produce the machine language interface. Because they are microcomputer specific, they are routinely interactive and screen directed. They tend to emphasize abstraction, and present the logical, vice physical view of the database and the hardware. Additionally, many DBMS command languages can access and execute non-DBMS command language programs e.g. assembly language utility programs and they possess limited ability to read non-DBMS data files. Perhaps the most valuable component of these languages is their effectiveness per line of the code. Drawing upon the internal DBMS functions and the structure of the database, a single line in a DBMS command language e.g. a file initialization procedure can

have the same effect as 50 to 100 lines of HOL source code. Understandably, this macro multiplier characteristic has received positive response from the marketplace. A final point is that microcomputer DBMS command languages bear a strong resemblance to their mainframe contemporaries. This can serve to decrease programmer retraining in some situations.

Despite these definite advantages, DBMS command languages are not perfect. Mirroring the database structure, they are often intrinsically limited by this structure. This can serve to constrain the number of fields per record, and records per database. Also, the number of databases that can be simultaneously accessed can be too few to support the desired application. More importantly, the rampant popularity of DBMS and related command languages has spawned a cottage industry of 'Ma and Pa' systems houses. As with any new industry, the reliability and robustness of their products does not always correspond to the advertised results.

As mentioned, the applications of DBMS command languages are widespread. Perhaps the most impressive example of broad acceptance of such languages is Ashton Tate's dBASE II.[1] Independent market analysts estimate this product

---

[1] dBASE II is a registered trademark of Ashton Tate.

accounts for 70% of the microcomputer DBMS market [Ref. 15 : pg. 114]. Using dBASE II as a parent database manager, the following systems written in dBASE II command language are currently operational: [Ref. 16 : pp. 138 - 141]

1. A restaurant management system that performs inventory, sales, payroll and invoicing functions. Developed by a part owner of a restaurant chain, it is currently utilized in 20 locations within the chain.

2. A student attendance system that provides real time information on pupil absence and tardiness, and produces corresponding reports. Users enjoy the improvement it represents over the previous system, and like the way it tracks student attendance patterns.

3. A beauty shop management system that performs billing, collects sales data, aggregates cost accounting data and maintains general ledger accounts.

4. A political campaign management system that tracks contributions and expenditures, does personnel management tasks for the volunteer campaign workforce, and provides prioritized appearance scheduling for the candidate.

5. A medical system that collects information on infections developed by incoming hospital patients, does an analysis of the possible sources of the infection, and recommends a course of action to locate, prevent and eliminate the infection. Users estimate this system produced annual savings of $150,000 during the first year of operation.

Simulation languages are special purpose languages. As the name implies, such languages have the explicit objective of simulation. Such languages trace their origins to mainframe computers and the science of mathematics. This family of languages has undergone an iterative and divergent development process. This development process has resulted

in the lack of strong commonality among modern simulation languages. Regardless, the following characteristics are identifiable:

1. Emphasis of abstract data types.

2. The language directly reflects the nature of the conscious designdecisions directed toward specific simulation techniques i.e. deterministic simulation, stochastic simulation, discrete simulation, continuous simulation.

3. Simulation languages generally have a strong interface with system design languages. This relationship is a managerial attempt to reduce the complexity of simulation.

4. Routinely, simulation languages have been derived from an existing programming language. This infers all the advantages and disadvantages of the derivative source are potentially present in the simulation language. This is not necessarily the case.

As the preceding characteristics infer, simulation languages are very flexible programming tools. They contain simulation development lifecycle aids, especially in the design area. Additionally, their integral design and structure support simulation of real world events in a more comprehensive manner than the other outlined software alternatives. Lastly, they can access a myriad of non-simulation language programs to produce a multitude of results.

As might be expected, the price of simulation language flexibility and power is high. Normally, a simulation language will imply a 16 bit microprocessor, due to the increased RAM requirements of the language. Additionally,

simulation languages tend to require instruction sets that are not always present on common microprocessors. This has led microcomputer simulation languages to lack the programming richness of their mainframe relatives.

IBM's General Purpose System Simulation (GPSS) is a popular simulation language. Likewise, SIMSCRIPT is the other predominate simulation language. GPSS relies on a macro assembler for execution, while SIMSCRIPT is FORTRAN-based. A profusion of applications, ranging from cash register queuing models to complex environmental impact models, are written in simulation languages. Academic institutions tend to be heavy users of simulation languages, as they provide an appropriate environment to deal with the topics of higher mathematics and operations research.

D. ALTERNATIVE EVALUATION AND SOFTWARE SELECTION USING THE SOFTWARE DECISION MODEL

Using previously described methods, the software decision model was exercised. The results of the model are displayed in Appendix I.

As the resulting mean software requirement values indicate, the microcomputer DBMS command languages represent the preferred alternative. Subsequent discussion with the project manager and the user group concerning the model outcome and software preference resulted in across the board

agreement on the use of a microcomputer DBMS command language for the simulation.

The designation of a DBMS command language as the generic software package to employ led the development team to investigate and research current market product offerings in this realm. A two week market review ensued, and resulted in the recommendation of the dBASE II command language as the specific software product to procure. Again, a discussion of this recommendation with the project team followed. The discussion resulted in unanimous concurrence with the recommendation.

The specific project rationale for selection of dBASE II as the software component of the simulation were:

1. Based upon the previousdefinition of conventional microcomputer technology, dBASE II would not bias or constrain the hardware decision by being processor or operating system specific.

2. The expansive list of applications examples of dBASE II, both in the academic and commercial world, provided a compelling argument to the project team as to the capability of dBASE II for the intended application.

3. As the model indicated, the features of dBASE II tend to be directly and positively related to the software requirements definitions. Additionally, this relationship was strengthened by the presence of relational database architecture. This firm relationship served as another convincing reason to select dBASE II.

4. dBASE II receives strong vendor and aftermarket support, and is readily commercially available.

5. dBASE II is fully transportable in the conventional microcomputer technology environment, and produces a

supportive environment to apply the principles of software engineering.

With regard to the procurement of the commercial product, the process consisted of the following steps:

1. A request, and a continuation sheet (SF 36) were forwarded to the Purchasing and Contracting office. This request contained a justification of the need, the suggested retail price, and a recommended vendor.

2. Based upon the suggested retail price, the Purchasing and Contracting office made several phone contacts with vendors concerning the product. This produced a list of price offerings from the vendors, which provided the vendor selection decision criteria.

3. The Purchasing and Contract office prepared the required contractual documents, and acquired the product from the vendor. Likewise, these documents entered into the appropriate financial systems for obligation and expenditure purposes.

V. HARDWARE

A. INTRODUCTION

One of the basic objectives of the PEP microcomputer simulation was to parallel the lifecycle development process of a major system . Adjustments in scale and scope were made, in accordance with thesis guidance, to achieve this objective. Appropriately, the system lifecycle process was analyzed to identify the hardware related phases of the process. As a result of this research, it was determined that the integration of hardware into a major system project may be described in four distinct phases:

1. Selection and Evaluation.
2. Acquisition.
3. Implementation.
4. Maintenance.

The above phases were used as a guide for merging PEP microcomputer simulation hardware with application software to forge a complete working system.

This chapter will examine each of the preceding hardware system development phases, first by discussing the principles of theory that are associated with each phase, and second by providing the development team's experiences with the execution of each phase. This approach was used

to provide a basis of comparison between academic theory, and what is commonly referred to as "real world" encounters.

B. HARDWARE SELECTION AND EVALUATION

Many consider that selecting the right computer hardware to support a given application is the most critical element for a successful completion of a major system's project. While the validity of this statement is debatable, it is certainly true that making the right system hardware choice, in the initial system design stage, will have a tremendous impact on the success of the project. This impact can be measured in terms of man-hours allocated to the effort, eventual effectiveness of the implemented system and total system lifecycle cost.

The hardware selection/evaluation phase can best be defined as a process by which alternative hardware configurations are examined for suitability, with respect to measurable evaluation criteria. A single hardware mix, representing the optimal solution within problem constraints is thus defined.

This definition provides a lead-in to the first principle of hardware selection/evaluation:

1. Principle: Regardless of the methodology that is used to select a particular hardware configuration, a quantifiable set of evaluation criteria must be developed as a means of establishing proof of hardware correctness for a given application.

The danger in violating this principle is that hardware selection will be based on some arbitrary set of evaluation criteria, which may or may not have any relevance to solving the problem at hand. This situation often occurs when personal bias is introduced into the hardware selection decision by the system developer, the user, or a combination of the two. This bias can manifest itself in numerous ways. There is the "leading edge of technology" bias that eliminates all hardware proposals which do no represent the state-of-the-art in technological advancement. There is the "I want the biggest and the best" bias that does not recognize the overhead costs associated with maintaining under utilized capacity or seldomly used features. There is the "strength in numbers" bias that bases system hardware selection on what everyone else has without considering the unique aspects of the problem that has to be solved.

The potential consequences of violating the quantifiable results principle are:

1. Iterative system design concepts of validation- Are we building the right system?, and validation- Are we building the system right? are not supported.

2. If the hardware does not perform as intended, it is almost impossible to go back and determine what the original rationale was for selecting the hardware in the first place.

3. Perhaps the most important consequence is that correcting hardware selection mistakes after the hardware has been acquired and put on line is generally very expensive. This may require the user to change the original scope of the project, or absorb

the opportunity overhead costs that are generated due to the system's lack of expected effectiveness.

Logically following the quantified results principle is the usage oriented principle:

2. Principle: Hardware selection should be based on its ability to support a specific application(s).

The risk of user dissatisfaction with selected hardware will tend to increase if the hardware is chosen to support a wide spectrum of applications when the nature of the requirements indicate that specialization is required. Violating this principle can result in loss of overall system efficiency. This type of loss is normally the result of cost minimization. Through this lowest cost mindset, the amortized cost savings frequently are less than the cumulative cost of lost system efficiency. This scenario can result when hardware is selected as a universal cure-all for every application. The reality is that every decision implies tradeoffs. Very often, general purpose hardware is not capable of performing specific tasks with the degree of efficiency that will satisfy the needs of every user group. An example of this general versus specific trade off is evidenced when a general purpose computer, commonly used in business applications, is employed in a scientific environment. Although the general purpose machine is capable of executing scientific applications, there is an incremental transfer cost in terms of efficiency that may be

greater than the incremental benefits that are realized when the machine is used to perform both functions. It may be perfectly acceptable in certain instances to select hardware to be used in support of multiple applications. However, the system developer needs to be aware of the advantages and disadvantages that result from this course of action.

Change is a fundamental variable in any environment, as reflected in the following hardware principle:

3. Principle: Selected hardware should be flexible enough to adapt to predictable changes in application scope.

In keeping with the analogy of death and taxes, it is nearly certain that once hardware has been selected for a given set of application requirements, those requirements will grow to require additional hardware resources. If the selected hardware cannot be expanded incrementally to satisfy these new applications demands, one of two possibilities will result. Either the new demands will go unsatisfied, which can result in user frustration and dissatisfaction, or a completely new system will have to be purchased to encompass both the old and the incremental requirements. This approach is usually cost prohibitive, based on the original system's lifecycle cost. The validity of this principle is evidenced by the fact that most market conscious microcomputer hardware vendors sell products with physical and logical expansion slots.

C. HARDWARE SELECTION/EVALUATION DEVELOPMENT TEAM ADAPTATION

The three preceding principles directly influenced the decision making process used to select the hardware component of the PEP microcomputer simulation. The first principle to be invoked was that of defining the application the hardware would have to support. Before any real hardware alternatives were discussed in any formal sense, the exact nature and scope of the simulation was obtained from the thesis advisor and the user's group. This information was developed iteratively so that all parties concerned, the thesis advisor, the user's group, and the development team, could be assured that the objectives of the PEP simulation were clearly defined and understood by all. Once the simulation objectives had been finalized, they were transformed by the development team members, in conjunction with the user's group, into a more detailed set of objectives referred to as a functional specifications. As the decision to use dBASE II as the software component of the simulation had been made, the following software-driven minimum hardware requirements were produced;

1. RAM: 64 KB for 8 bit processors, 128 KB for 16 bit processors.
2. Disk Drives: Two, to allow for segregation of applications and systems aspects of the simulation.
3. Disk Capacity: At a minimum, 200 KB per formatted disk.

4. Operating System: Previously defined within the conventional microcomputer technology philosophy.

These functional specifications were used to invoke the quantifiable evaluation criteria principle. With complete definition of the simulation objectives and functional specifications, the development team was able to identify and rationally evaluate hardware configurations from different vendors. A two dimensional matrix was chosen as the basic evaluation format for the different hardware products. The first evaluation step was to compare each of the hardware alternatives from a common reference point. This point was determined to be the mix of hardware that would satisfy the mandatory requirements of the functional specifications. These mandatory hardware requirements included:

1. Screen Size: 80 columns by 24 lines. This was deemed essential, as the parent system was implemented on terminals with these characteristics. The user group provided this requirement.

2. Monitor Size: 9 inches diagonally, as a minimum. This was included to consider eye fatigue and readability of the text in the simulation. Again, the user group provided this requirement.

3. Total Hardware Cost less than $3000.00 per unit: This reflected the resource constraints of the project sponsor, and the fact that four stand alone simulation microcomputers were desired for the MCPCC.

4. Local Maintenance Support: This requirement was designed to minimize the risk of not having the deliverables ready for the December 1983 MCPCC. It was driven by the development team.

Next, the total cost for each vendor's particular hardware configuration was calculated. The matrix was designed to evaluate different vendor's hardware configurations based on both quantitative and qualitative data. The quantitative data was intended to provide a barometer of hardware performance in terms of capacity i.e. disk storage, main memory, monitor sizes. The qualitative data was used to access the relative benefits of the different features associated with each vendor's hardware. These different features fell into the category of desirables, and included the following:

1. Bundled Software: This was considered, as it offered the opportunity to explore other potential applications for this hardware outside the realm of this thesis.

2. Graphics Capability: While not essential, this feature was included due to its value in a teaching environment and in management communication.

3. Numeric Keypad: This item was included, as it would reduce data entry problems during training sessions.

4. Designed as a Portable: This was included as a desirable because of the anticipated out of the classroom use of the simulation.

5. Detachable Keyboard: Again, this was included to enhance the ergonomics of the machine.

6. Hard Disk Capable: This was included due to the inherent speed of a hard disk, and due to the anticipated system program and file size.

7. RAM Expandability: Again, this was included due to the processing speed enhancement potential that it represented.

A subjective weighting system was used to convert desirable feature characteristics to numerical values. These values would serve as the evaluation mechanism. The underlying strategy for using this approach was to calculate a hardware performance to price comparative index for each hardware configuration. This index could then be used to determine which hardware configuration represented the best value for each dollar expended. Further, it would permit a hierarchical ranking of alternative hardware configurations. Although portability was specified as a hardware constraint, configurations that were not specifically designed as portable were considered in the evaluation process. Rationale for inclusion was to determine if any significant tradeoffs involved with the use of portable hardware as compared to desktop systems.

The development team gathered the necessary data for different vendor's hardware offerings and their associated market cost by conducting personal visits to established retail outlets, and by researching appropriate trade journals. Once that data was collected for each hardware system, it was run through the evaluation matrix. As a result, the KAYPRO 10 was found to have the highest performance-price index and was consequently selected as the most desirable hardware system to support the PEP

microcomputer simulation. The evaluation matrix and the rankings of the other systems are displayed in Appendix D.

D. ACQUISITION

After all the hardware alternatives had been evaluated, and a particular configuration was selected, it had to be be obtained from a vendor(s). Acquisition can be defined as the process by which an organization secures delivery of equipment and material from a vendor for use in a previously defined end application. [Ref. 17: pg. 154] This generic definition is applicable in describing both public and private sector acquisition processes. However, the remaining discussion of hardware acquisition will emphasize the process employed by federal agencies and the military services.

The major criticism that has been directed at the Federal Government's acquisition process is that it is unnecessarily complicated. As a result, it is often considered to be non responsive to user procurement needs. Procurement delays that are caused by a cumbersome and inefficient acquisition process can have a significant impact on the success or failure of a systems project. This is especially germane with ADP equipment, due to the rate of technological change. Yesterday's competitive edge and productivity coup can turn into today's opportunity loss if enough time transpires between the project initiation and

actual receipt of equipment. The acquisition process the Federal Government uses for major ADP procurements has been consistently rebuffed for the time that elapses between formal need identification and equipment receipt. This excessive delivery time lag often results in equipment that is technically obsolete by the time of receipt.

One of the primary reasons the Federal Government's acquisition process is so awkward is that not only does one have to comply with a bureaucratic hierarchy of rules and regulations within a given department, but these rules tend to overlap between different departments i.e. fiscal and supply. Successful completion of a procurement cycle, from the formal identification of a requirement to the actual receipt of the the equipment, involves interfacing with two separate staff communities: finance and supply. Each of these communities have their own regulations that deal with material procurement.

The acquisition process involves the commitment of financial resources, and the services of the contracting branch of the supply department. The process normally commences with the formal identification of a new initiative, stated in monetary terms by a POM (Program Objective Memorandum) submission. The POM is used as a long range planning document which serves to highlight the fact that proposed special projects will require funding if

approved. The financial phase of the acquisition process continues, given project approval, by incorporating the previously identified project costs in the budget submission. The project may be given a separate line item as in the case of a major system development, or might be included as a subset of more general classifications of financial resource requirements. Regardless of the method used to identify project funding requirements, the budget is the vehicle that is used to show the planned allocation of financial resources, for a given appropriation type, by major classification of expense, for the life of the appropriation. If the project survives the budget review process, funds are subsequently made available for obligation purposes. The process from POM submission to actual project funding can span as much as three years or more.

The contracting portion of the acquisition process commences once the project funds have been identified and made available for obligation. The contracting portion of the acquisition cycle typically receives the most criticism with respect to procurement inefficiencies. This criticism is from the fact that the contracting office that must comply with all the rules and regulations that govern equipment procurement. These rules and regulations can be quite complex, depending on the magnitude of the acquisition

project. The magnitude is usually determined by the life cycle cost of the procurement. In the category of ADP, certain procurement rules have received a considerable amount of notoriety such as the Brooks Act, OMB Circular A-109, GSA (General Services Administration) directives, and numerous other military peculiar regulations which are listed in the Defense Acquisition Regulations (DAR).

It became apparent, as the federal government's acquisition process was researched, that two principles could be developed to positively influence an acquisition from start to finish. These principles were identified as follows:

1. Principle: Start the acquisition process as soon as possible.
2. Principle: Once the acquisition process has been started, insure that the project is continually monitored, through each phase of the process, and initiate corrective action when necessary.

E. ACQUISITION - DEVELOPMENT TEAM ADAPTATION

The development team realized that a careful and comprehensive acquisition strategy was needed to successfully procure the selected PEP simulation hardware. The resulting acquisition strategy was based on the aforementioned principles.

The first part of the strategy involved converting system hardware requirements into estimated funding

requirements. PEP simulation hardware cost estimates were obtained by researching the current market price for each line item of the selected configuration, and then by extending the price for the identified line item quantity. Each extended line item price was then totaled to determine the system cost of the PEP hardware procurement. This budget estimate was forwarded to the project sponsor for review and approval. The intent of the entire PEP budget formulation and review process was to identify and obtain a source of project funds. It was assumed that the O & M,MC appropriation would be used to fund the project vice PMC due to the original scope constraint that a given line item cost not exceed $3000.00, and the established project completion date. The Accounting Branch, Code FDA, Fiscal Department, Headquarters, Marine Corps agreed to support the PEP simulation project with funding equal to the costs that were identified in the budget estimate. This funding authority made it possible to proceed into the next phase of the acquisition process.

Armed with appropriation data, the development team was able to initiate the actual procurement of the desired hardware configuration. This involved a considerable amount of interface with the NPS purchasing and contracting office, which is a division of the supply department. The appropriate requisition forms were obtained, and the

designated hardware was submitted for procurement action. The total cost of the requested hardware, while under the investment appropriation threshold, was beyond the threshold for sole source procurement. This is where the development team got its first experience with the bidding process that is used promote competition. Three vendors submitted bids for the requested material, and, in keeping with the lowest cost to the government philosophy , the lowest bidder was awarded the primary contract. A second vendor was selected to provide a system development prototype machine. The primary contract involved the procurement of four KAYPRO 10 computers and designated peripheral devices. The system prototype contract involved the procurement of one KAYPRO II machine and its peripheral devices. The KAYPRO II was chosen as the prototype development machine because it was rated second to the KAYPRO 10 in the hardware decision matrix. The duration of the entire procurement process was seven months, commencing in July 1983. Completion occurred in March, 1984.

The second principle of systems acquisition was applied extensively during this time frame. A considerable amount of time was spent by the development team conducting follow up procurement action to ensure that the prescribed delivery dates for the hardware were met. The success of the PEP simulation project was intimately linked to having the

requested hardware available for the systems development during the July - November 1983 period. There was very little slack for project delays due to hardware unavailability. This was driven by the proximity of the required software completion date to the MCPCC implementation date. Continual procurement followup action by the development team was absolutely essential to having the hardware delivered at the right time, in the right quantities, and in right condition.

F. IMPLEMENTATION

This is an extremely sensitive phase of the system life cycle. Despite the quality of preceding work, a poor implementation can lead to the death of an otherwise healthy project. Implementation can be defined as the transition of a system from a conceptual design into a physical and concrete working system [Ref18 : pg. 78]. Implementation normally involves two integrated yet distinct components 1) hardware, and 2) software. An implementation strategy will normally determine how these two components will be managed with regard to their time phased introduction as a working system. In some cases, hardware and software are treated as separate entities, despite the fact they are intended to function together as a system. This is governed primarily by the amount of complexity that is associated with each of the respective portions of the system, and the projected

organizational impact of the system. For large, complex systems, it is often necessary to initially accomplish the hardware implementation first and get it functioning correctly. Next, the software implementation is addressed as a separate process. This type of situation occurs normally when major system projects are initiated. Human beings are only capable of effectively dealing with a limited amount of complexity, as measured by the number of events that require simultaneous management action. Often a system development which involves the concurrent implementation of both hardware and software is simply too complicated for one person, or group of people, to effectively manage. This sort of complexity is normally dealt with by breaking up the problem into smaller, more manageable pieces. In the case of a major system, this involves splitting the hardware and software implementations into separate, related management efforts.

The implementation strategy for less complex systems is markedly different from complicated systems. It is usually possible to manage the simultaneous integration of hardware and software. This is often the case when the software does not have to be developed without precedent, is commercially available, or does not represent a complicated programming effort. This type of system can be brought on line as an

integrated package more readily as many of the encountered implementation problems can be managed with relative ease.

Although a system integration can involve both hardware and software issues, this discussion will focus on the hardware related issues. From a hardware perspective, there are usually three management problems that effect the hardware implementation: 1) file conversion, 2) parallel operations, and 3) people, i.e. user involvement. Each of these issues will be addressed separately.

Most complex systems must interact with or be overlaid by existing data files. When new hardware is introduced from a different manufacturer than the existing system, it is almost inevitable that some type of file conversion process will be required. This can be a very risky, time consuming, and expensive process depending on the degree of vendor incompatibility that exists, and the size and quantity of the files requiring conversion. Software conversion costs can play an important part in the vendor decision. For instance, even if a vendor clearly manufactures a superior product in terms of computing power relative to the organization's current equipment, software conversion costs might be sufficiently high as to negate any potential benefits that might be enjoyed by using the new vendor's equipment. Consequently, an organization will tend to remain with one hardware manufacturer even if the product

that manufacturer produces is less capable than other market offerings. This approach is a simple plan to avoid the cost of software conversion. This is a routine decision consideration in private industry. The Federal Government does not consider these costs to the same degree, in the interest of fostering increased hardware competition in the market place.

In addition to the cost of converting software, there is the technical risk associated with the conversion process. Many manufacturers provide utility programs that help with the conversion process. However, the reality of the situation is that software conversion from one hardware type to another often does not proceed as smoothly as might be advertised. The resulting consequence of this is frequently that the user must correct conversion problems using in-house personnel. This usually translates directly into unbudgeted project costs. There is nothing more disheartening than pressing the button to run the new system, after the conversion process has supposedly been completed, only to get an error message back from the computer such as "CAN'T READ FILE".

One of the most difficult aspects of hardware implementation is developing a strategy to accomplish a smooth and cost effective transition from the old hardware to the new hardware. There is a delicate balance that must

be struck between a cautious but costly approach to system phase in, and switching to the new system before it has been proven to be operationally ready. The cost of running dual systems can be significant, and every realistic effort should be made to minimize this cost. However, the consequences of abandoning an old system and switching to a new system before it is capable of handling an operational load can be devastating. This has especially been the case with many novice microcomputer implementations. Experiences abound where a vital business function, such as accounting, was converted to a new computer system before it was proven to be reliable. In the worst cases, no recovery provisions had been made to use the former system in the event the new system failed catastrophically. It would seem evident that if there was any question as to where the fulcrum of the balance should be put, it would be better to incur some inefficiency costs due to unnecessary parallel operations, rather than risk a complete collapse of the function all together due to system debugging problems.

People, as with most management related issues, play a very important part in the over all success or failure of a major system initiative. In fact, no matter how technically eloquent a given system is, it will be the people i.e. the users of the system that will determine if the initiative succeeds or fails. There have been many

documented cases where a given system project looked good on paper, but, as a result of poor implementation procedures, failed to gain acceptance within the organization and went unused or was under utilized. The implementation mistake that is often made with hardware is that systems designers, who are themselves very familiar with the hardware, fail to recognize that the users may not have had any prior experience with the hardware. Users need information to understand what the intended function of the hardware is, and how it is supposed to be used. Terms that the systems designer may be very comfortable with such as disk packs, channels, main memory, and graphics terminal may be totally foreign to the end user of the hardware. The user may speak in terms of cash flow, stress points, outstanding TAD, which may be alien to the systems designers. A successful hardware implementation requires that users be interfaced with system developers during the implementation process. Under ideal conditions, this interface would be accomplished by establishing comprehensive training programs that users would be required to attend. This training would be supplemented by documentation in the form of a users manual, written in response to the user's need for information. If it is not possible to train the users prior to the formal implementation of the hardware, then the users documentation must be particularly good to insure users understand what

the hardware is supposed to do and how to use the hardware. Often, systems designers will write the users documentation. The problem with this approach is that system designers frequently introduce their hardware familiarity bias into users documentation manuals and tend to make false assumptions about the level of expertise of the target audience. As a result of this bias, they fail to provide the information the users need in the proper format. The consequences of this are that users will:

1. Get frustrated and not use the system.
2. Use the system, but will not take advantage of its full capabilities.
3. Try and come up with procedures based on trial and error.
4. Obtain less sophisticated but understandable hardware that may not be as capable as the system hardware. The current boom in microcomputers on manager's desk, while a large mainframe sits in the Data Processing department, is a prime illustration.

Many of these problems could be avoided if users are allowed to actively participate in the hardware implementation process.

G. IMPLEMENTATION - DEVELOPMENT TEAM ADAPTATION

The primary and secondary contract items were received in accordance with the delivery schedules of the respective contracts. The first item of hardware to be used directly in the PEP microcomputer simulation project was the prototype KAYPRO II machine. The machine was used to

develop a quick working prototype of the actual PEP microcomputer simulation project. This equipment was received at the end of July, and work was immediately begun on the PEP prototype software development. There were no parallel operation considerations to deal with as the focus of the implementation was primarily to become familiar with the hardware, and to obtain preliminary system performance data in terms of main memory, off line storage requirements, and system response times. The short term implementation goal of the prototype machine was to have a working skeleton of the actual PEP simulation available for a demonstration for the August, 1983 MCPCC. It was also intended to test the portability aspect of the hardware by taking the prototype system to a test site at Camp Pendleton, California, where the actual PEP system was being installed on mainframe hardware. This trip allowed the development team to evaluate the robustness of the PEP simulation hardware, and provided an important opportunity to get an eyewitness look at how the parent system functioned. More information concerning the software implications of the trip will be provided in the chapter covering system development.

The primary hardware items, four KAYPRO 10 computers, were delivered in September, 1983. This equipment, while still part of the same family of hardware as the prototype

machine, represented an entirely different architecture in terms of execution speed, off line storage formatting, and disk capacity. This hardware was intended to execute the actual PEP microcomputer simulation software, and form the nucleus of the project.

The KAYPRO 10 hardware was intended to support two primary functions. First, it had to be capable of supporting the software development effort of the PEP simulation. Second, it had to be able to run the PEP application in an actual class room environment. The KAYPRO 10 hardware was able to support the PEP software development process much better than the prototype machine for two reasons: 1) the KAYPRO 10 represented a much more capable hardware configuration than the prototype machine in terms of its ability to provide an efficient interface to software development tools such as text editors, and other applications programs, and 2) many of the lessons that were learned using the prototype hardware were directly applicable to the new PEP hardware.

The actual operational implementation of the KAYPRO 10 hardware, was scheduled to occur during the December, 1983 MCPCC. The goal was to have the entire PEP software development completed by this time. The success of this implementation, and indeed the thesis project itself, hinged on the development team's ability to employ the second

principle of hardware implementation, getting the users involved in the process. The strategy that was adopted to insure this user involvement took place was to have the users group actually write the User's Manual for the December MCPCC PEP demonstration. The user's group was provided with the documentation about the KAYPRO 10 hardware that was supplied by the vendor. Their task was to take this information and repackage it so that the MCPCC students would have adequate information to use the simulation. Having the user's group write the PEP User's Manual was considered an appropriate means to offset familiarity bias, an implementation pitfall that the development team consciously sought to avoid.

A survey was taken after the the December, 1983 MCPCC implementation of the PEP simulation to obtain the students evaluation of the hardware's effectiveness, and the manner in which it was implemented. In general, the results of the survey were very favorable on both accounts, and indicated that the hardware implementation had gone very well. More detailed student evaluation results will be provided in a later chapter, which addresses all of the PEP project findings.

H. MAINTENANCE

This phase of the hardware lifecycle normally does not get the attention that it deserves, yet it can prove to be

one of the most expensive elements of a system's lifecycle costs. Maintenance can be defined as the process by which any degradation in hardware performance from variances in system specifications, to complete failure, are corrected [Ref 18: pg. 196]. It is certain that once hardware is put into service, it will require varying degrees of maintenance support over time. The only way for an organization to insure that it does not get caught by surprise when the repair truck shows up, or fails to show up, is to have a well planned hardware maintenance strategy in place that is preemptive vice reactive. The specific provisions of a maintenance contract will vary from organization to organization, depending on how critical a role the hardware plays in the daily operations of the firm. In some industries, such as banking, computer down time that is caused by hardware problems can have a devastating impact on the organization. Millions of dollars can be held in queue as transactions go unprocessed. Organizations that rely on real time processing for their life blood should have a maintenance contract that supports this type of operation. A computer manufacturer has even gone so far as to base their entire business on the importance of hardware reliability and low maintenance requirements. Tandem Computer's market niche is that they sell computers which are extremely reliable due to built in redundancy. Hence,

the equipment failure occurrences that will require major maintenance, and involve extended down time, are negligible. Many organizations have purchased Tandem computers because of these features. Other types of organizations, which rely primarily on batch processing to satisfy their ADP needs, may be able to tolerate nominal amounts of down time due to hardware failures. However, even when batch processing operations are involved, there must be some type formal agreement as to who is responsible for what type of maintenance i.e. preventive, corrective, at what echelon, and what constitutes an acceptable amount of down time before it begins to adversely affect the organization.

I. MAINTENANCE - DEVELOPMENT TEAM ADAPTATION.

The maintenance strategy formulated by the development team was to ensure that vendors who were awarded contracts would be capable of: (1) providing maintenance support beyond the manufacturer's standard equipment warranty, (2) ensure that a technical support facility was geographically close enough to resolve major or minor maintenance problems within twenty four hours, or (3) loaner equipment would be available for this contingency, and (4) that manufacturer promulgated revisions to the basic hardware would be provided by the vendor at no cost to the Government. The short run goal of this strategy was to minimize the risk of missing the designated December PEP implementation date due

to hardware related delays. In the long run, it was designed to prevent protracted maintenance problems in future applications.

There has been occasion to test the robustness of the maintenance provisions of the hardware support contract for both minor and major maintenance problems. The maintenance strategy appears to be sound, based on the development team's preliminary experience in this area. All problems that were encountered, subsequent to the receipt of the hardware, were resolved without incident. All maintenance work was completed within contract specified timeframes.

VI. THE SIMULATION DEVELOPMENT LIFECYCLE

A. INTRODUCTION

The purpose of this chapter is to demonstrate the integration of the simulation development lifecycle into this thesis. This demonstration will focus on the "How", not the "What". It is not intended to be a redundant discussion of the simulation development lifecycle. Rather, it is a descriptive narration of the creation and development of the PEP microcomputer simulation. To structure this narrative, the chapter sections will commence with the simulation feasibility phase and proceed through the simulation maintenance phase.

Before continuing, it is appropriate to list the primary supporting documentation that was employed in this simulation. This supporting documentation included:

1. The PRIME Enhancement Project User's Manual (Draft Copy): Published by the PEP development team, this document provided the simulation development team with a comprehensive system summary, user operating procedures for data input and queries, and general background information on the parent system.

2. The PRIME Enhancement Project Supervisor's Manual (Draft Copy): Also published by the PEP development team, it provided the simulation development team with a comprehensive summary of PEP's supervisor subsystem. Further discussion of this subsystem is presented in section C of this chapter, The Simulation Requirements and Specifications Phase.

3. United States Marine Corps Order 7300.10B, Mechanized Financial Procedures for Selected Marine Corps Posts and Stations: This document provided file, record and field definitions for PRIME. As such, it was a ready-made data dictionary for use with the dBASE II command language.

4. dBASE II Assembly Language Relational Database Management System: The user's manual for dBASE II, this document provided the development team with a wealth of language-specific information. It is a product of Ashton Tate.

B. THE SIMULATION FEASIBILITY PHASE

As the name implies, this phase dealt with the achievability of the simulation. From a managerial perspective, the feasibility analysis is viewable in three subsets: simulation feasibility, economic feasibility and technical feasibility.

The simulation feasibility question is the focus of this thesis. As such, this feasibility subset will be addressed in chapter VIII, Conclusions and Recommendations. During the actual simulation development lifecycle, the simulation feasibility was assumed to be doable. This assumption motivated research into the matter and propelled the remainder of the simulation effort.

A high degree of overlap existed between the previously described software and hardware requirements (chapter IV and V), and the technical and economic feasibility. This relationship was compelled by the ability to define mandatory software and hardware requirements, and the

subsequent identification of candidates that met the defined requirements. This connection implied the technical feasibility of the simulation. Further, the integration of a mandatory cost ceiling into the requirements definition and the availability of candidates within the cost ceiling implied economic feasibility. In summary, the fact that there were candidates that conformed to the mandatory software and hardware requirements was interpreted by the project team as a positive technical and economic feasibility.

Normally, the macro definition of the parent system to simulate is done in the simulation feasibility phase. As outlined in chapter III, the PRIME Enhancement Project was designated as the parent system to simulate.

A cost/benefit analysis is routinely accomplished in this phase of the lifecycle. In this project, a cost/benefit analysis was not formally performed. Rationale for this omission was:

1. The fact that a project sponsor was willing to commit financial resources to a research effort indicated a higher management decision that expected benefits exceeded the anticipated costs. Relying on this barometer, the performance of a cost/benefit analysis was judged to be a redundant task.

2. The fact that the project was driven by research objectives, vice concrete, quantifiable output measures, tended to make the performance of a cost/benefit analysis an amorphous exercise.

3. The degree of financial resources committed were minimal, with respect to the perceived benefits at the project team and project sponsor level.

Concluding the simulation feasibility phase, the project team conducted a schedule analysis and performed some preliminary project development. Often, these tasks are accomplished at later points in the simulation development lifecycle. They were done at this point in this project to minimize project risk by establishing milestones. The outcomes of the schedule analysis and preliminary project development were:

1. The formation of the user group that served to finalize the project team composition. Further discussion of this subject can be found in chapter III.

2. The August,1983 MCPCC was designated as the fielding date for the prototype system. The December, 1983 MCPCC was set as the implementation date for the simulation.

3. The user group formulated a training scenario revolving around budgeting and an electronic spreadsheet. As this was outside the scope of the defined simulation, the user group assumed the complete responsibility for development and implementation of the package.

C. THE SIMULATION REQUIREMENTS AND SPECIFICATION PHASE

The purpose of this phase was to designate the essential elements of the parent system that were to be present in the simulation. The software and hardware requirements were defined concurrently, and are presented in chapters IV and V, respectively.

To designate what to draw out of the parent system, the project team conducted a comprehensive review of the PEP User's manual. This self-education process was followed by a liaison visit to the Accounting Office, Marine Corps Base, Camp Pendleton, California. At this site, the parent system was operational on an Amdahl 470/V7 computer. Additionally, this visit allowed the simulation project team to interface with the PEP development team.

The trip to Camp Pendleton produced the following results:

1. The PEP development team informed the simulation development team that another primary subsystem of PEP had been implemented. Known as the Supervisor subsystem, it had grown out of recent user requests and a surge development effort. A piece of primary documentation, the PEP Supervisor manual, was obtained.

2. Response times to interactive PEP users on the parent system were observed to be slow and cumbersome. The assigned operating system priority of PEP, and high processor utilization were discovered as the explanation of this slow response time.

3. A complete source listing of PEP, written in NATURAL, was obtained.

4. There was a general consensus among PEP users that PEP was preferred over the previous alternative, a SCANDATA product.

Upon completion of the liaison trip, the learning process of the simulation project team had advanced enough to allow an intelligent and meaningful definition of the simulation requirements and specifications. This effort produced the following:

1. The parent system employs interactive input to a batch transaction file. This feature of the parent system would not be present in the simulation. Rather, the simulation would employ the identical interactive input mode coupled with on-line file updating. This decision reflected the fact that the simulation was a mono-user system, and a batch file would be inappropriate for this specific environment. Also, on-line file updating corresponded to the available user training periods. It was further felt that an on-line file update mode would avoid the slow response times observed on the parent system, and better hold the student's attention in a training environment.

2. The following parent system input transactions and inquiry files were designated as essential elements to be present in the simulation:

   a. Input transactions: AD, AOMIL, AOCIV, AOMIL, A2CIV, BU, and RECAP. These transactions effect personnel records in the Personnel History File (PHF).

   b. Inquiry files: The Unfilled Orders file (UFO), Reimbursable Order Number file (RON), Personnel History File (PHF), and Master Job Order Number file (MJON).

   In the user group's opinion, these were the most frequently used transactions and files in the parent system. As such, they would be vital to the simulation and tend to enhance the training process. The remaining transactions were not functionally developed by the development team due to time limitations, and with an eye toward future development efforts in other thesis projects. The omitted transactions would be physically present in the simulation, however, they would only be output screen formats and would not accept interactive inputs.

3. The LOGON and LOGOFF procedures in the parent system would not be in the simulation. They contained a communications protocol, and non-English oriented prompts and responses. The LOGON and LOGOFF system in the simulation would be a simplified abstraction of the parent system designed to demonstrate security features and emphasize menu-driven responses. Additionally, the LOGOFF procedure would incorporate certain hardware-specific hard disk considerations.

4. Transaction and inquiry screen formats and edit checks were drawn directly from the PEP User's manual. File definitions and structures were drawn from Marine Corps Order 7300.10B. The underlying simulation data manipulation and processing was abstracted from development during this phase. It eventually became the responsibility of the development team, and tended to be a function of the dBASE II command language.

5. An initial effectiveness indicator, a user feedback survey, was planned and developed.

6. It was formally decided that the Supervisor subsystem would be simulated only in a text driven, screen format. It would not accept user data input. Again, this decision reflected the time limitations of the development team. Also, the Supervisor subsystem was an order of magnitude more complex, in programming and design terms, than the Input and Inquiry subsystems. As such, it would have required doubling of the development team personnel to implement. These resources were not readily available. Nor was it desirable to add to the project team at this phase in the simulation development lifecycle.

At this point, the simulation requirements were placed in the documentation file. This manual file provided a project team database to support documentation efforts.

D. THE CONCEPTUAL SIMULATION DESIGN PHASE

This phase was concurrent with the previously discussed hardware and software acquisition processes. The first step in this phase was the development of a fundamental model of PEP. Appendix C contains this model. User data entry is the only source of input. This was driven by the previous requirement for an interactive environment. Four possible outcomes, corresponding to the different simulation states, were further identified. The discrete simulation states

reflected the requirement for a menu-driven system. This model was reviewed by the user group. They concurred with its content.

Progressing to a more defined stage, a hierarchical process diagram was developed. It is portrayed in Appendix D. Functionally dividing the diagram into modules, the following modules were conceptualized:

1. The Master Control Module: This module was the operating system and database management interface with the hardware, contained the LOGON and LOGOFF modules, served to implement security features and performed system state determination.

2. The Input Transaction Subsystem: This subsystem is a menu-driven module designed to permit a simulation user to input transactions. It has a direct interface with the database monitor to perform interactive edit checking of input data. Likewise, the interface with the database monitor is the link to the on-line updating of databases. The subsystem has an internal hierarchy of different transaction types.

3. The Inquiry Subsystem: This subsystem is a menu driven module that allows a simulation user to designate the desired file and file view. The interface with the database monitor accesses the database and executes the desired query. The file view is definable to the field level, and includes aggregate and individual record views.

4. The Supervisor Subsystem: This subsystem is a menu driven module that permits supervision of the Input and Inquiry transaction subsystems. It interfaces with the database monitor to achieve this supervision. Inherent in this subsystem is the ability to adjust simulation edit checks and error trapping defaults.

5. The Unexpected Events Handler: This module operates at the system level. It includes the peripheral surge protector, the hardware reset button, file initialization routines and a hard disk 'crash proofing' routine. It is designed to counter the normal realm of calamity.

6. The Database Monitor: This module is the simulation kernel. It contains the database management system, and serves as the coordinator of the entire simulation execution. It interprets data input, manipulates databases, interprets and causes program execution, and performs simulation housekeeping.

7. The Edit Validation Database: This module implements edit checking on the data inputs to the Input Transaction subsystem. Database structure is drawn from the PEP User's manual.

8. The Inquiry Validation Database: This module implements error trapping on the Inquiry Transaction subsystem. Database structure is taken from the PEP User's manual.

9. The Supervisor Validation Database: This module is the error detection device for the Supervisor subsystem. Database structure is drawn directly from the PEP Supervisor's manual.

This hierarchical process diagram provided a solid foundation for prototype development. Using the KAYPRO II hardware, the prototype was demonstrated on schedule at the August, 1983 MCPCC. The prototype demonstration surfaced the following:

1. The development team was made aware of an open file limitation of dBASE II. Also, the development team recognized a recursive call programming problem. While problems are never desired, this situation was a blessing in disguise. It made the development team rethink some of the fundamental program interface and calling methods. This change, made apparent by the prototype demonstration, caused the development team to save a significant amount of time and frustration in the detailed simulation design phase and in the simulation module creation phase.

2. The prototype concretely demonstrated the need for the KAYPRO IO due to access speeds, and floppy diskette storage limits.

During this phase, the user group was formally designated as responsible for the preparation and development of the simulation user's manual. Coupling the users to the simulation in this manner produced stronger testing, and a more understandable, easier to follow simulation user's manual.

The phase concluded with a management review of the project. Requirements validation and milestone evaluation were included in the review. Detailed transaction module schedules and testing periods were identified. Tighter controls were applied as the project drew closer to the implementation deadline.

E. THE DETAILED SIMULATION DESIGN PHASE

This phase began with further decomposition of the hierarchical process diagram. This decomposition produced individual program modules. These individual program modules were defined from the following parameters:

1. The User's perspective: The screen format.
2. Data Structure and Databases: Reflective of the PEP User's manual (edit checks and error trapping), and Marine Corps Order 7300.10B.
3. Interface Relationships: Designation of senior-subordinate module relationships, module calling and data flow to and from the module.
4. Module Purpose: Functional specification of the module's primary task(s).

A detailed breakdown of the above is contained in the PEP

Microcomputer Simulation Systems Reference Manual, which is discussed later in this chapter. Appendix E contains a summary of this decomposition by subsystem and program module.

This phase covered everything short of actual program coding. It was a time intensive process, and served to demonstrate to the development team the critical need for planning in the simulation development lifecycle.

F. THE SIMULATION MODULE CREATION PHASE

This phase involved taking the products of the detailed simulation design phase, and transforming them into dBASE II command files. These command files contain the executable dBASE II statements that breath life into the simulation. The time and effort invested in the detailed simulation design phase was well spent as the transformation process went fairly smoothly. This was also a reflection of the many English-like functions in dBASE II.

Faced with limited personnel resources, the two members of the development team became specialists. One member assumed the responsibility for coding the edit-intensive Input Transaction subsystem. The other took on the challenge of the Master Control module and the Inquiry Transaction subsystem.

Throughout this phase, the development team relied on the following programming support resources:

1. WordStar Word Processing System: This MicroPro product was employed as a text editor. Also, it allowed command file execution directly from the text editor environment.

2. The dBASE II User's Manual: This voluminous manual is often criticized for its complexity. The development team found it to be a useful and understandable document. However, they felt it was intended for use by at least moderately experienced programmers.

3. The KAYPRO 10 microcomputer: The integral 10 MB hard disk, coupled with WordStar, provided a superlative development and debugging bed.

4. Epson dot matrix printers: Their speed and output readability were essential to program development. Also, they enhanced the documentation effort.

In addition to command file development, the development team created the requisite databases and initialized the database records. This process was interfaced with the user group tutorial development effort to ensure system consistency.

Realizing that programs written by others are frequently difficult to understand, the development team applied documentation standards to the command files. Each command file had a header with the following entries:

1. File purpose.
2. Parameters passed to the file.
3. Parameters passed from the file and where to.
4. Error checking.
5. Programmer.
6. Date the file was last updated.

Within the program, each major function was highlighted with

a descriptive comment, and appropriately spaced to delineate the function. The command files were reviewed by the user group for understandability, and suggested documentation improvements.

Unit testing, which receives an in-depth discussion in the next chapter, was performed in this phase. After a development team member produced a 'debugged' program, it was presented to the user group for a 'road test'. This procedure enhanced the user group interface with the simulation. It also removed the program from the author, who could have been unconsciously avoiding invoking program errors. The entire process provided good feedback, and served to establish a near real time verification and validation cycle.

G. THE SIMULATION MODULE INTEGRATION PHASE

Upon completion of unit testing, the program modules were integrated into the simulation. This was concurrent with the previous simulation development lifecycle phase. Interface testing and variable passing were examined in this integration. Also, this phase served to validate error trapping and edit checking performed by the databases across the system. The objective of this phase was a validation/verification/refinement cycle, coupled with a corresponding documentation updating. The reader is

directed to chapter VII for a detailed description of this process.

The next phase was the acid test: implementation. To prevent any surprises, a management review was held at the conclusion of the simulation module integration phase. Consistency of the tutorial and the simulation were stressed. Also, the milestone attainment was examined.

H. THE SIMULATION IMPLEMENTATION PHASE

The simulation was fielded at the December, 1983 MCPCC. It included the following:

1. One hour of in-class familarization and system overview. Tutorials were distributed to the students. The objective of this session was to introduce the students to the simulation and its equipment, and to minimize student resistance to computer use.

2. The students each took a KAYPRO 10 to their quarters after class hours. Here, they worked through the tutorial. The event provided good informal feedback on the simulation to the project team.

3. Upon completion of the MCPCC, the user feedback survey was administered. The results are discussed in chapter VII.

Redundancy was the key to the simulation backup strategy. The strategy consisted of the following layers of backup:

1. Primary Data Location: User area A0, KAYPRO 10 hard disk.

2. Secondary Data Location: Floppy diskettes held by the development team.

3. Tertiary Data Location: NPS computer center, under user identification number 0250P.

4. Tertiary Data Location: Hard copy printout held by the thesis advisor and the second reader.

While not 100% fault proof, this strategy is capable of preventing most data loss catastrophes.

The development team's documentation, the PEP Microcomputer Simulation System Reference Manual, was developed from the project documentation file. In addition to a complete listing of command files, the following areas are contained in the manual:

1. Overall System Configuration.
2. System Hardware Configuration.
3. System Software Configuration.
   a. Commercial software.
   b. Development Team Software.
      i. PEP.
      ii. The speadsheet scenario.
4. System Backup.

Permanent distribution of this document was to the thesis advisor and second reader. This distribution is intended to permit intelligent future development of the simulation, based on a solid presentation of previous development efforts and technical information.

This phase concluded with the user group refining the simulation user's manual on the basis of the user feedback survey.

I. THE SIMULATION MAINTENANCE PHASE

Currently, the primary components of this phase are:

1. Use of software utilities e.g. FINDBAD for minor hardware preventive maintenance.
2. Increased source code documentation.
3. Direct interface with the vendor for hardware corrective maintenance.

The future maintenance strategy will be a direct function of the development path pursued with this simulation. Alternative development paths are discussed in chapter VIII.

VII. TEST AND EVALUATION

A. INTRODUCTION

This chapter will present the topic of test and evaluation. This topic is a vital part of the system development process as it typically determines how expensive system maintenance will be. Research data indicates that system maintenance costs have historically represented the largest portion of a system's lifecycle costs [Ref: 11: p.18].

This chapter will be developed in accordance with the underlying structure of this thesis. First the principles of test and evaluation will be discussed. These principles are: (1) formal and informal testing procedures, (2) the relationship of requirement specifications to testing and quality assurance, (3) quality characteristics, (4) quality metrics, (5) automated test and evaluation tools, and (6) the life-cycle implications of testing and evaluation. Secondly, the development team's implementation of these test and evaluation principles will be discussed in the context of tangible application results.

B. BACKGROUND

Prior to studying an abstract topic, it is useful to define the key terms and concepts that are germane to the

topic. This procedure was applied to the topic of test and evaluation. This ensured that frequently used words were clearly defined, relative to the topic.

The point of entry for the identification of terms can begin with the subject title itself, "test and evaluation". These two words are frequently used together to represent a single evolution. Although these words are similar, they should actually be decomposed into separate pieces to describe the actual processes that occur. System testing is concerned with the reliability of a given set of hardware and software, respective to a given application. The word "reliability", as it is used in the preceding statement, refers to three characteristics: [Ref. 19: p. 3-14]

1. Accuracy of computed results.
2. Compliance with system specifications.
3. Robustness.

Evaluation is concerned with how much, or to what degree, a system satisfies some pre-determined set of ideal performance measures [Ref. 19: p. xi]. Thus, the evaluation process permits a system to be judged, and perhaps ranked, on its ability to satisfy the application requirements. The similarity between test and evaluation is that both processes are used to document a system's performance and behavior. The primary difference between the two processes is that testing is concerned more with the details of system

correctness and is more specific than evaluation. Evaluation typically considers more aspects of system performance other than simply system reliability.

Testing and evaluation can be conducted at different levels within a system hierarchy. There are two basic levels: the systems level and the component level. For clarity, it is important to specify which level is being referenced. Test and evaluation criteria at the component level will be oriented toward a more detailed perspective than at the system level. Although individual components usually are integrated to make a complete system, the test and evaluation criteria and procedures that are used at each level can be quite different. This discussion will encompass both levels and will be explicitly differentiated when appropriate.

C. FORMAL AND INFORMAL TESTING PROCEDURES

The primary objective of any system developer is to create a "quality" end product in terms of both hardware and software. This section will examine the characteristics which define software "quality". Research indicates that it has been particularly difficult to measure the quality of software in a working system [Ref. 19: p. xvi]. This problem is no small matter as software has been shown to be the dominate cost factor in the lifecycle of most major systems [Ref 11: p. 30 - 32].

The PEP simulation development team recognized how important software quality would be to the success of the project and consequently formulated a test and evaluation strategy that would support the design of quality software. The first issue of the test and evaluation strategy that had to be resolved was to determine what type of testing method to use: formal or informal? Formal testing was found to be most appropriate for situations where the scope of software complexity was low to moderate [Ref. 20: p. 80]. Formal testing procedures are used to establish the mathematical proof of correctness of an algorithm in accordance with some previously defined set of specifications. Formal testing is typically only used in conjunction with small software modules. Rationale for this is that as the size of the software increases, so does the complexity of the procedures that are used to establish its proof of correctness [Ref 20: p. 76]. The complexity of formal testing can be driven to the point where it is no longer feasible to conduct such testing. This is due to the current lack of automated tools to perform formal testing, and the fact that the testing methodology itself may contain errors if the procedures that are used become too complicated.

The PEP microcomputer simulation was reviewed to determine to what degree formal testing would contribute to achieving the objective of producing reliable software. As

a result of this review, it was determined that formal testing was not well suited for this particular application. Next, the alternative method of informal testing was then examined for its suitability with respect to software testing and reliability.

Informal testing was found to be synonymous with the conventional notions of software testing such as debugging and interface input/output checks. Informal testing is routinely used in situations where complex systems are involved, or where a large number of integrated modules must be tested [Ref. 20: p. 80]. Informal testing is used in these cases because there are automated tools available that can assist with the testing process, and the test itself is not as rigorous a proof as in the case of formal testing. On this basis, the PEP microcomputer development team decided that informal testing was the best method to use for the software reliability analysis.

It should be stressed that informal testing is not to be confused with imprecise testing or inaccurate testing. In fact, informal testing can be considered quite "formal" in the context of precision when it is used in conjunction with software reliability design enhancements such as structured programming, modularity, and formal specification requirements. The relationship of these design concepts to

informal testing will be explored in greater detail in this chapter.

It should be noted that the choice to use informal testing, as opposed to formal or piecemeal testing, was made to increase the probability that the delivered PEP simulation software would be reliable. Even if software is exhaustively tested it cannot be considered 100% "error free". This is because testing only proves the presence of errors, it can not prove their absence [Ref. 20: p. 76]. A vendor's claim that a given software product is completely without bugs because it successfully passed a testing procedure should be considered wishful thinking.

The testing process should be structurally supported from the outset of system development by an appropriate design structure. The PEP microcomputer simulation development team considered two software testing structures: "top-down" and "bottom-up". The decision to use one design alternative over the other was governed by how well each of the respective alternatives would support follow-on testing. The top-down design method was found to be the generally better approach. Top-down design permits system testing on an 'as you go' basis, because higher level modules are developed and tested before lower level modules. Also, it is not necessary to have all the lower level modules completed prior to higher level module testing. Calls to

incomplete lower level modules are replaced by program stubs. Using this technique, interface output parameters of the higher level modules can be tested for accuracy. Bottom-up testing relies on the lower level modules being developed first. These individual modules are tested and then integrated into the overall system. The major problem with bottom-up testing is that the module integration phase is frequently rough and choppy [Ref. 18: p. 159]. Also, even though individual modules can be tested, it is difficult to test the system as an entity until all the subordinate modules have been completed and integrated into the system. At that point, if program bugs are encountered (usually at module interfaces), they are very difficult and very costly to correct.

An added benefit of using the top-down design approach was testing by the users early in the simulation development lifecycle. The nature of top-down design initially required construction of a complete skeleton of the PEP system. This was the master control module (chapter VI) in the hierarchy, and it controlled the flow of control to all the subordinate modules. This permitted the users to test the working PEP skeleton for errors, despite the fact that none of the lower level modules had been developed.

Getting the users involved in the testing process at such an early stage in the software development was

invaluable to the overall success of the project. The combination of top-down design and user involvement in the testing process yielded numerous benefits. It set the stage for future module testing, the users felt as if they had a vested interest in the software design, and most importantly, system design and development errors were isolated early enough to take corrective action without much difficulty. An example of this testing methodology's utility was evidenced when the users requested that a change be made to the PEP skeleton. The requested change modified a module calling sequence from the design specifications. It was effected without difficulty primarily because the actual interfaces of the lower modules that would have been effected by this change were not yet in place. Had these modules been integrated into the PEP system before the requested change was made, it would have been significantly more difficult to make the desired change. This 'test as you go' procedure with the users group was was used throughout the remainder of the PEP simulation development lifecycle.

D. REQUIREMENTS SPECIFICATIONS, TESTING AND QUALITY ASSURANCE

In order to perform any type of testing there must be a set of criteria that is available as a test standard. These test criteria are frequently referred to as requirements specifications. These specifications, whether at the

systems level or the component level, should provide a clear statement of what the software is intended to do. At the component or unit level, requirement specifications often state test boundary ranges, input/output parameter values, and module interface consistency. They tend to be identified in significantly more detail than requirement specifications at the system level.

As described above, each module was developed in a top-down hierarchical precedence. After each module had been developed, the following tests were performed:

1. It was tested for data accuracy if computations were involved.
2. Data input/output validity.
3. Compliance with requirement specifications, both formal and informal.
4. Robustness.

The testing process was accomplished on a completely manual basis. Extensive error handling routines were built into modules that were vulnerable to erroneous input conditions or system states. Robustness of individual modules and the system was a primary design consideration. It reflected the the diverse nature of the group that would be using the system. Robustness was tested in two ways. Known error conditions would be introduced into the module to test and exercise its error handling capabilities. The second method was to pass development team debugged modules to the users

for a robustness 'road test.' Experience with this technique showed that if it were possible for an error condition to be produced, the users would find a way to make it happen. This provided a very effective means of identifying where error handling routines needed reinforcement, and where to put self checking input edit routines. Input edit routines were used extensively as a method of achieving enhanced robustness.

The degree of formality of the testing methodology is directly influenced by the formality of the requirements specifications [Ref. 19: p. xxix]. These specifications can range from simple verbal instructions to formal automated requirements definitions. The degree of formality usually mirrors the size and complexity of the software development project. With respect to testing, requirements specifications provide a point of reference that can be used to determine if a system is being built correctly. A typical problem that is encountered with software requirement specifications is their oftentime ambiguity in terms of precisely defining what the software is supposed to do. Recent developments in the automation of software requirements definition have greatly enhanced the testing process by alleviating much of the ambiguity problem [Ref. 19: p.xxix]. However, these automated tools are normally economical only when used to support large and complex

software development projects. Semi-structured, written English requirement specifications are still widely used for projects of small to moderate complexity. The problem with English language requirement specifications is that their tendency toward ambiguity. It should be apparent that the testing process is adversely affected by the ambiguity that can result when common English statements are used to describe requirement specifications. The reason that formal automated requirement specification techniques are not used to support small and moderate projects is due to the processing and personnel overhead costs associated with such methods.

It is envisioned that the cost of these types of programs will decrease as their use becomes more widespread. More detailed information about these formal requirement specification language methods will be provided in a latter section of this chapter.

The primary documents used to define the requirement specifications of the PEP simulation were considered formal in the sense that the displays and figures were reasonably precise in their definition of what the parent system's purpose. The ambiguity problem was manifest when the users attempted to state their requirements verbally using the English language. One of the PEP simulation original scope constraints was the entire PEP parent system could not and

would not be emulated in the timeframe alloted for the project. The users group was responsible for identifying the exact bounds of this scope constraint. They were to develop a clear and concise statement of system requirement specifications, reflective of any deviations from the actual PEP User's manual. The user's group requirements specifications were imprecise, difficult to follow, and difficult to test against. This situation should be avoided at all costs. The problem was eventually solved by using an iterative requirement specification refinement process between the PEP simulation development team and the users group until an acceptable set of requirement specifications was produced.

E. CHARACTERISTICS OF QUALITY SOFTWARE

The predominate share of the preceding discussion on software quality has been presented in terms of software reliability. This is certainly an important characteristic of software quality. However, it is not the only attribute. Research has indicated that the quality of a particular software product can be described in terms of the following disjoint characteristics: [Ref. 19: p. xi]

1. Understandability
2. Completeness
3. Conciseness
4. Portability

5. Consistency
6. Maintainability
7. Testability
8. Usability
9. Reliability
10. Structuredness
11. Efficiency

The relative importance of each of these attributes is function of user requirement specifications and the application.

The PEP simulation team took conscious measures to ensure that each of the above quality characteristics was incorporated into the structure of the system design. The criteria used in the software evaluation matrix provide a tangible example of this deliberate software quality design effort. The rationale for using a particular software evaluation criteria was based on the attributes of software quality that were identified above. The decision to use dBASE II as the implementation language for the PEP simulation reflected the development team's concern for quality software production. dBASE II readily supported modern software design principles like structured programming, top-down design, modularity, and self documentation. These qualities were exploited in the detailed design and module creation phases of the PEP

simulation development lifecycle with one singular goal in mind: the production of a quality product.

F. QUALITY METRICS

The term "metric" is defined as a measure of the extent or degree to which a product possesses and exhibits certain characteristics [Ref. 19: p. xv]. The focus of this chapter has been on the quality characteristics of software. However, the scope of the discussion need not be restricted to this narrow context. It is the user who ultimately determines what set of attributes should be present for a product to be considered a "quality" product. Once these attributes have been defined, quantifiable metrics can be developed to evaluate the degree to which a given characteristic is present. Theoretically, if the chosen characteristics accurately represent the quality of a product and the metrics determine to what extent these characteristics are present, then a high correlation between the characteristic and the metric should indicate that a product is indeed a "quality" piece of work. The method used by the PEP simulation development team to evaluate the software quality in the simulation system will be discussed in a separate section of this chapter.

G. AUTOMATED TOOLS THAT AID TESTING AND EVALUATION

System and software design is an abstract process which

involves the transformation of creative ideas into concrete requirement specifications. Ambiguity is often generated as an unwanted by-product of the transformation process due to the manner in which humans deal with complexity. As previously stated, ambiguity, in terms of requirement specifications, seriously impairs the process of system testing. Based on this realization, it is apparent that a need exists for design tools which can help humans automatically define precise requirement specifications from abstract ideas. Such automated design tools have recently been developed which are intended to accomplish this objective.

Automated support tools were developed primarily to satisfy the software design and implementation needs of major system projects. The complexity of these systems, measured in terms of requirement specification volume, simply became too difficult to manage using conventional verification and testing techniques. The Ballistic Missile Defense (BMD) system is an example that is often used to illustrate this phenomenon. The Defense and Space Systems Group of the TRW Corporation was responsible for the BMD project development. The requirements document for the BMD system is comprised of no less than 8248 individual requirement and support paragraphs, and is contained in a 2,500 page specification document [Ref. 20: p. 160]. TRW's

solution to this management nightmare resulted in the creation of an automated software design support tool called SREM (Software Requirements Engineering Methodology). The purpose of SREM was to allow the system designer to state requirements in a reasonable natural language and then use automated techniques to handle the details of keeping track of changes, ensuring consistency, and reporting the iterative effects of statements [Ref. 20: p. 160]. A special requirement specification language and database model processor were developed to implement SREM. The special language was called Requirements Statement Language (RSL). This language used precise syntax to define requirement specifications. The use of this syntax accomplished two things. First, it helped alleviate the requirement specification ambiguity problem. Second, it allowed a regular interface to permit computer processing of RSL defined requirement specifications. The Abstract System Semantic Model (ASSM) is a relational data base that is used to store RSL defined specifications. This data base is an integral part of the SREM system. The actual application programs used in the SREM system to perform requirement specification management use the ASSM for data retrieval and file storage.

Other types of automatic design tools have been developed which take a slightly different approach than

SREM. SADT (Structured Analysis and Design Technique) was also developed to help designers write clear and precise requirement specifications and automatically perform data base oriented house keeping functions. The primary difference between SADT and SREM is that SADT makes extensive use of graphics to describe the interaction of system modules and requirement specifications. Problem Statement Language/Problem Statement Analyzer (PSL/PSA) is another example of an automated requirement specification management tool.

Although there may be slight implementation differences between different automated tool types, the more important issue to consider is what are the common characteristics of the automated tools. These common denominators are:

1. Singularity of purpose.
2. Structured syntax for requirement specification language definition.
3. Routines to translate tool defined requirement specifications into a computer processable format.
4. Custom data bases to store information.
5. Application programs which manipulate the data that is contained in the data bases.
6. A function oriented user interface i.e (graphical, text oriented or a combination of both).

The PEP simulation development team did not use an automated requirement specification tool in the system design or test phase for three primary reasons:

1. No such tools were available at NPS either on the mainframe or the VAX minicomputer.

2. Such tools do not currently exist for use on microcomputers.

3. The scope of the simulation was not considered complex enough to warrant the use of these types of tools even if they had been available.

The rationale for the preceding in depth discussion of automated tools was to demonstrate that even though an automated requirement specification methodology was not used herein, the conceptual foundation of these tools is relevant to any system design project. As such, their use should be a decision point when a system design strategy is formulated.

H. LIFECYCLE IMPLICATIONS OF TESTING AND EVALUATION

This chapter has examined many of the principles that are involved with testing and evaluation. The unanswered question at this point in the discussion is to what extent does testing and evaluation influence the success or failure of a systems design project over its lifecycle? The answer to this question should provide some insight into the rationale of why a separate thesis chapter was devoted to this topic.

Testing and evaluation are used for two primary purposes:

1. To document system performance in accordance requirement specifications.

2. Establish the framework for follow-on system maintenance.

These functions are critical to a successful implementation of a system design project. System documentation of test and evaluation results has historically been abysmal [Ref. 12: p. 146]. System designers have typically viewed documentation as a necessary evil, rather than a useful reference instrument. As a result of this prevailing attitude, test and evaluation documentation is either:

1. not produced at all.
2. derived in a haphazard manner as a secondary consideration after the fact; or
3. not described in sufficient detail to be of any real use.

Test and evaluation documentation is frequently treated as the stepchild of system development because it does not provide the immediate results, and hence gratification, to system designers that working code does. However, it is this type of documentation that can be of invaluable assistance in the debugging process and system maintenance.

It is the system maintenance aspect of test and evaluation documentation that is really critical. It is virtually impossible to make systematic corrections or modifications to a system without the proper documentation. Test and evaluation documentation provide module interface information and system test results. This allows system modifications to be made in a rational manner. It is also provided so reasonable predictions can be made about the

system's future behavior after the modifications have been made. Given the proportion of system maintenance cost to the rest of the life-cycle cost, test and evaluation documentation should be assigned top priority in any major system design project. The test and evaluation documentation procedures employed by the PEP simulation development team are identified in the following section.

I. PEP SIMULATION TEST AND EVALUATION RESULTS

The PEP simulation development team decided to use a multi-faceted strategy for testing and evaluation. The strategy is described as follows:

1. Software testing will be initially segregated from the process of evaluation.
2. Software testing will be done initially at the component or unit level.
3. Informal testing methodology will be used.
4. The PEP User's Manual (Draft Copy), and user group input will define the domain of requirement specifications.
5. These requirement specifications will be used as the software test criteria.
6. Individual system modules will be tested in accordance with the above procedures.
7. Individual modules will be integrated into a comprehensive working system i.e the PEP simulation project.
8. The testing process will be merged with the evaluation process. The scope of testing will transition from details of component level to general system level.

9. Users evaluate the system using predefined system quality metrics.

10. Users evaluations and the software are the test and evaluation documentation.

The above test and evaluation strategy was considered successful as it provided structure and performance definition to the entire process.

As individual modules were completed they were tested by two groups, the development team and the users group. These modules were compared with the appropriate set of requirement specifications to ensure that such things as screen formats, computed results, and input data were implemented in accordance with the functional specifications. This process was conducted iteratively until any discovered discrepancies had been corrected. A master file was used to store and identify the most current versions of modified system modules.

The integration of the individual modules into the complete system was extremely easy due to the characteristics of top-down design. System integration was essentially an on going process as the higher level modules were developed and tested before the lower level modules. Completion of the system integration process allowed the transition to be made from software testing at the component level, to testing and evaluation at a higher and more general system level. This transition represented an

important phase of the test and evaluation process. This importance was due to the decision that, while the final version of the PEP simulation certainly could not have been completed without detailed module testing, the results of testing and evaluation at the systems level would provide a better barometer of how well the end product had actually been designed and built. This decision was predicated on the fact that users of the PEP simulation would not be concerned with the details of how the system was put together or functioned. Their primary interest would be how the system performed as an integrated entity. It was also decided that, since the PEP simulation was built for the users, the users should determine the "quality" of the end product. This decision was considered consistent with the principle of information hiding as the development team dealt with the testing details of module interaction. The users' responsibility for testing and evaluation would only be concerned with those aspects of the PEP simulation at the system level.

The concept of quality metrics was introduced at this point in the test and evaluation phase. The PEP development team quite logically tailored these metrics to the system level evaluation. These metrics were chosen to evaluate more than just software. Although this certainly was a major consideration, they were also designed to evaluate

hardware and system documentation. These metrics were incorporated into a questionnaire format so that users could make subjective appraisals on the extent that certain system quality characteristic were or were not present in the working version of the PEP simulation. The questionnaire was constructed in two parts. The first part was designed to obtain background information about the characteristics of the user group. The second part was intended to translate the opinions of the users' group, with respect to system quality, into quantifiable results. This translation was implemented by constructing a low to high evaluation scale for each selected quality metric. A high score indicated that the user felt a high degree of the quality attribute was present. Conversely, a low score would indicate that the system lacked this particular measure of quality. The user group background data was obtained to help explain the results of survey findings. The interpreted results of the survey were compiled to document the test and evaluation findings. This documentation was intended to be used as the reference source for system maintenance and future project enhancements.

An example of the questionnaire that was used is provided in Appendix F. The development team analyzed the survey results and drew the following conclusions:

1. The software and hardware performed in accordance with design specifications.

2. System documentation, in the form of the PEP Simulation Users manual, was well written and useful in terms of system operation.

3. The PEP simulation provided a reasonable degree of correlation with the actual PEP system.

4. Users considered their participation in the PEP simulation implementation to be a valuable learning experience.

5. Adequate feedback information was obtained from the users, in the form of recommended changes to the PEP simulation, to make future revisions to the current system.

The test and evaluation phase of the PEP simulation was considered successful in terms of detailed component level testing and system testing. The development of a detailed test and and evaluation strategy early in the design of the PEP simulation life-cycle was considered an essential part of its successful implementation. The use of quality metrics and the other principles of testing and evaluation presented in this chapter were also considered to be important parts of the system design effort.

VIII. CONCLUSIONS AND RECOMMENDATIONS

A. INTRODUCTION

This chapter has three distinct purposes. First, it will provide responses to the research questions outlined in chapter III. Next, it will develop and support conclusions from the research and application efforts related to this thesis. It will conclude by recommending future courses of action to pursue with the simulation effort.

This chapter is written from a dual perspective: the managerial viewpoint and at the systems level. The concept is to take the macro approach to the topics.

B. RESEARCH QUESTION RESPONSES

The research questions originally posed in chapter III, have provided a research oriented direction and emphasis for much of the development and application effort. Additionally, they have a practical application to the organizational management of AIS, microcomputers and simulation.

1. The first research question was:

   Are microcomputers technically capable of executing software packages that simulate a major AIS?

The author's response to this inquiry is yes. Rationale for this response is:

a. Many of the software packages used in microcomputers give the hardware a multiplier effect. This capability enables the hardware to handle tasks normally assumed to be performed by larger machines. For example, the dBASE II use of a relational database manager that performs functions with a single command line that might take 50 lines of BASIC to perform i.e. a database sort.

b. The mono-user nature of most microcomputer systems allows them to concentrate their processing power on one primary task. This is relative to the mainframe environment, where most AIS's exist, that supports multiple processing and multiple programming. The primary mission of the microcomputer is to run the simulation. As such, it can devote all of its resources to accomplish the task without distraction.

c. Many of the AIS's were developed for mainframes with less processing power and less efficient software than are available on today's microcomputers. This type of situation favors the simulation of an AIS on a microcomputer.

2. The second research question asked:

What are the positive and negative benefits associated with simulating a mainframe AIS on a microcomputer?

To respond to this question, the thesis authors relied on their development experience, the experience of the users at the December,1983 MCPCC, their educational backgrounds, and their personal experiences in the military.

a. The following positive benefits were identified:

(1) The microcomputer presents a less costly environment in which to perform simulations, relative to mainframe and minicomputer alternatives. From a managerial perspective, this is the primary advantage of this approach to simulation.

(2) Generally, microcomputers support simulation development better than mainframes and minicomputers. This flows from the fact that

microcomputers are usually mono-user systems, and the developer does not have to compete for processing resources. Also, some of the development tools, such as text editors. on microcomputers are much easier to use than their mainframe counterparts.

(3) Microcomputers, because of their widespread availability and their novelty value to certain people, tend to encourage the use of simulations. Also, portable microcomputers like the KAYPRO family make the simulation transportable to the user. This removes the mandatory trip to the terminal room or use of a modem required to execute a mainframe simulation. This effect is significant in a field training environment.

b. The following disadvantages were identified:

(1) A simulation is extremely sensitive to the simulation requirements and specifications phase. This sensitivity flows from the fact that the objectives and project plan are clearly developed in this phase. Any weakness in this process can have a ripple effect throughout the remainder of the simulation development lifecycle. This becomes critical in the microcomputer arena, where there is a lack of sophisticated software requirements and development tools. While their mainframe relatives have techniques such as SREM and SADT, microcomputers still rely heavily on traditional development methods i. e. flowcharting. This can generate a pitfall as these techniques may be insufficient to handle the complexity implied in a simulation. As such, an inherent disadvantage to simulations on microcomputers exists. It will likely be resolved, as the microcomputer applications software industry matures and moves farther into the market growth stage of the product lifecycle.

(2) A simulation on a microcomputer is that it leaves a microcomputer in an organizational setting. Any employee, with a little imagination, can think of many alternative uses of this device. This can inject change into an organization, which can generate both positive and negative effects. From a negative

perspective, it can cause workers to pursue tasks other than their assigned positions i.e. developing applications, and it can violate corporate information policy. These factors must be realized when considering implementing a microcomputer-based simulation.

(3) The microcomputer simulation of a mainframe AIS implies that other computer resources exist in an organization. The presence of a microcomputer can create a redundancy of computing resources, and raises the question of effectiveness and efficiency of the other computing resources. Additionally, the problems of data compatability, telecommunications, and processor interface between the mainframe and the microcomputer can grow out of this situation.

(4) A microcomputer simulation can generate all the organizational negative effects previously discussed in chapter II.

3. Question 3, dealing with the specifics of simulation, asked:

What are the essential features that must be present in an AIS simulation to ensure it is a feasible and realistic model?

Reviewing the development cycle and the user's feelings, the thesis authors isolated the following features:

a. The user interface must be very similar to the parent system. Defined in simple terms, the screen formats must be the same. This creates the initial user interface with and perception of the simulation. It is where the user relies on past knowledge of the parent system to operate the simulation. Differences at this point would tend to discourage and confuse users. This type of frustration and dissatisfaction normally results in lack of user acceptance of a system. Lack of user acceptance is the first step to system failure, as the system has failed to meet the needs of the audience it was designed to serve.

b. Accuracy of simulation outputs and products is

vital. Lack of accuracy tends to hurt the simulation's credibility and causes users to revert to experimenting with the parent system. Additionally, lack of accuracy can serve to form an invalid training base for simulation users to apply when operating the parent system.

c. Simulation response time and turnaround must be equal to or better than the parent system. This feature exercises the simulation's time compression capability. Also, this time compression tends to better hold the user's interest and attention.

d. The support environment must be as transparent as possible to the user. This transparency decreases the complexity of the simulation from the user's perception. While increasing the ease of use, it also serves to minimize external distractions to the user. This feature was a consideration in the selection of the KAYPRO 10 over the KAYPRO II, as the KAYPRO II would have required the user to handle floppy diskettes to initiate the simulation.

The remaining spectrum of features and elements from the parent system can be abstracted, suppressed, or be subject to information hiding in the simulation. Their explicit absence does not to detract from the feasibility and realism of the simulation. Also, specifying this realm of features as essential would tend to drive the simulation in the direction of an emulation or replication.

4. The next research question dealt with attempting to introduce objectivity into a subjective environment:

> How can an AIS simulation be designed to maximize user satisfaction within organizational and technological constraints?

The first step in accomplishing this user satisfaction objective is to get the users involved in the simulation development effort at an early stage. Since they will be

3.) Have you ever used a personal computer:

A. At home	YES	NO
# OF RESPONSES	6	14
% OF SAMPLE	30	70

B. At work	YES	NO
# OF RESPONSES	13	7
% OF SAMPLE	65	35

4.) In response to the following questions, circle the number that best indicates your feelings toward the statement.

A. The tutorial was easy to use and understand.

	1	2	3	4	5
	Strongly Disagree	Somewhat Disagree	No Opinion	Somewhat Agree	Strongly Agree
# OF RESPONSES	0	0	0	8	12
% OF SAMPLE	0	0	0	40	60

Mean Response Value for Question 4A: 4.6.

B. The tutorial provided adequate guidance to allow me to enter information to PEP.

	1	2	3	4	5
	Strongly Disagree	Somewhat Disagree	No Opinion	Somewhat Agree	Strongly Agree
# OF RESPONSES	0	0	2	3	15
% OF SAMPLE	0	0	10	15	75

Mean Response Value for Question 4B: 4.65.

C. The tutorial was too long.

	1	2	3	4	5
	Strongly Disagree	Somewhat Disagree	No Opinion	Somewhat Agree	Strongly Agree
# OF RESPONSES	3	4	6	7	0
% OF SAMPLE	15	20	30	35	0

Mean Response Value for Question 4C: 2.85.

D. I was able to read the computer screen without any trouble.

	1	2	3	4	5
	Strongly Disagree	Somewhat Disagree	No Opinion	Somewhat Agree	Strongly Agree
# OF RESPONSES	0	0	0	9	11
% OF SAMPLE	0	0	0	45	55

Mean Response Value for Question 4D: 4.55.

E. The screen prompts for PEP were easy to follow.

	1	2	3	4	5
	Strongly Disagree	Somewhat Disagree	No Opinion	Somewhat Agree	Strongly Agree
# OF RESPONSES	0	1	0	11	8
% OF SAMPLE	0	5	0	55	40

Mean Response Value for Question 4E: 4.30.

F. I was able to follow what was happening on the screen from the tutorial.

	1	2	3	4	5
	Strongly Disagree	Somewhat Disagree	No Opinion	Somewhat Agree	Strongly Agree
# OF RESPONSES	0	1	0	10	9
% OF SAMPLE	0	5	0	50	45

Mean Response Value for Question 4F: 4.35.

G. I learned something positive about personal computers from the PCC.

	1	2	3	4	5
	Strongly Disagree	Somewhat Disagree	No Opinion	Somewhat Agree	Strongly Agree
# OF RESPONSES	1	1	1	9	8
% OF SAMPLE	5	5	5	45	40

Mean Response Value for Question 4G: 4.1.

H. I would consider getting a personal computer for my work environment based upon the PCC.

	1	2	3	4	5
	Strongly Disagree	Somewhat Disagree	No Opinion	Somewhat Agree	Strongly Agree
# OF RESPONSES	1	3	5	3	8
% OF SAMPLE	5	15	25	15	40

Mean Response Value for Question 4H: 3.7.

I. I was able to understand the value of electronic spreadsheets based upon the PCC.

	1	2	3	4	5
	Strongly Disagree	Somewhat Disagree	No Opinion	Somewhat Agree	Strongly Agree
# OF RESPONSES	0	2	7	9	2
% OF SAMPLE	0	10	35	45	10

Mean Response Value for Question 4I: 3.55.

J. I was able to understand what PEP can do for my unit from the PCC.

	1	2	3	4	5
	Strongly Disagree	Somewhat Disagree	No Opinion	Somewhat Agree	Strongly Agree
# OF RESPONSES	0	2	7	9	2
% OF SAMPLE	0	10	35	45	10

Mean Response Value for Question 4J: 3.55.

LIST OF REFERENCES

1. Naisbitt, John, Megatrends, Warner, 1982.

2. C4 Division, Headquarters, Marine Corps, Marine Corps Information Systems Support Plan, FY84 - FY89, 1983.

3. Bobulinski, R. A., "ADP and the Manager : A Beginner's Guide", Armed Forces Comptroller, pp. 40 - 42, Summer, 1982.

4. "A New Era for Management", Business Week, pp. 50 - 76, April 25, 1983.

5. Holland, Robert, "Distributed Databases : Decisions and Implementation", Data Communications, pp. 42, May, 1982.

6. Polzak, L. A. , "Word Processing : Productivity Solution or Paperwork Shuffle?", Armed Forces Comptroller, pp. 24 - 29, May, 1980.

7. Lewis, F. C. Major, USMC, The Management Information Needs of the Activity Level Comptrollers within the Marine Corps, Master's Thesis, Naval Postgraduate School, March, 1983.

8. Shannon, R. E. , Systems Simulation : The Art and the Science, Prentice Hall, 1975.

9. Bork, R. H. Jr. , "The Technology of Illusion", Forbes, pp. 158 - 162, 27 February, 1984.

10. DeBord, W. A. and Siebel, J. D., "Training MIS Users through Simulation", Management Accounting, pp. 36 - 42, January, 1982.

11. Boehm, B. W., Software Engineering Economics, Prentice Hall, 1981.

12. Brooks, Frederick P. Jr., The Mythical Man Month, Addison Wesley, 1982.

13. Boehm, Barry W., Software Engineering: R & D Trends and Defense Needs, TRW Defense and Space Systems Group, Long Beach, California, 1982.

14. Gore, Marvin, and Stubbe, John, Elements of Systems Analysis, William C. Brown Company, 1983.

15. "ABC's of dBASE II", PC Magazine, pp. 114 - 124, February 7, 1984.

16. "The Many Faces of dBASE II", PC Magazine, pp. 136 - 141, February 7, 1984.

17. Sherman, Stanley N., Government Procurement Management, pp. 50 - 229, Woodcrafters, 1981.

18. Summerville, I., Software Engineering, pp. 38 - 235, Addison, 1982.

19. Boehm, Barry W. and others, Characteristics of Software Quality, North-Holland, 1978.

20. Freeman, Peter and Wasserman, Anthony I., Tutorial on Software Design Techniques, IEEE Computer Society, 1980.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	Curricular Office, Code 37 Computer Technology Naval Postgraduate School Monterey, California 93943	2
4.	USMC Representative, Code 0309 Naval Postgraduate School Monterey, California 93943	5
5.	Commandant of the Marine Corps Code C4 Headquarters, Marine Corps Washington, D. C. 20380	2
6.	Professor Daniel R. Dolk, Code 54Dk Naval Postgraduate School Monterey, California 93943	1
7.	Professor Norman Lyons, Code 54Lb Naval Postgraduate School Monterey, California 93943	1
8.	Captain Bradley Mercer, USAF Code 52Zi Naval Postgraduate School Monterey, California 93943	1
9.	Professor William Haga, Code 54Hj Naval Postgraduate School Monterey, California 93943	1
10.	Major Robert Cowen, USMC Code FDA Headquarters, Marine Corps Washington, D. C. 20380	1

11. Mr. Eugene Regan — 3
Code FDA
Headquarters, Marine Corps
Washington, D. C. 20380

12. Captain Michael E. O'Neil, USMC — 2
Company B
Headquarters Battalion
Repair Division
Marine Corps Logistics Base
Albany, Georgia 31704

13. Captain Keith V. Lockett, USMC — 2
Audio Visual Training Branch
Marine Corps Development and Education Command
Quantico, Virginia 22134

14. Ms. Alyce Austin, Code 0141 — 1
W. R. Church Computer Center
Naval Postgraduate School
Monterey, California 93943

15. Colonel James Updyke — 1
Code FDB
Headquarters, Marine Corps
Washington, D. C. 20380

16. Major Jack Larson — 1
Central Design and Processing Activity
Marine Corps Development and Education Command
Quantico, Virginia 22134

207149

Thesis
L7871 Lockett
c.1 The simulation of a major Automated Information System (AIS) on a microcomputer.

207149

Thesis
L7871 Lockett
c.1 The simulation of a major Automated Information System (AIS) on a microcomputer.

thesL7871
The simulation of a major Automated Info
3 2768 002 12601 3
DUDLEY KNOX LIBRARY